1999

Adaptive agents for information retrieval and data-driven knowledge discovery

Jihoon Yang
Iowa State University

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

Adaptive agents for information retrieval and data-driven knowledge discovery

by

Jihoon Yang

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Science

Major Professor: Vasant G. Honavar

Iowa State University

Ames, Iowa

1999

Graduate College
Iowa State University

This is to certify that the Doctoral dissertation of

Jihoon Yang

has met the dissertation requirements of Iowa State University

Signature was redacted for privacy.

Committee Member

Signature was redacted for privacy.

Committee Member

Signature was redacted for privacy.

Committee Member

Signature was redacted for privacy.

Committee Member

Signature was redacted for privacy.

Major Professor

Signature was redacted for privacy.

For the Major Program

Signature was redacted for privacy.

For the Graduate College

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to express my heart-felt gratitude to my advisor Dr. Honavar for his guidance during the entire period of my graduate studies at Iowa State University. As a research advisor, he encouraged and motivated my work by giving me invaluable, candid feedback. I thank him for being accessible always, for giving me free access to his books and material, and for having patience with my awkward explanations and slow progress. I am also very grateful for his help in writing a number of papers and for his supervision of this dissertation. He has been the exemplar of a researcher to me, and influenced me in choosing my career. I also thank him for his considerate help and guidance when I was a teaching assistant in some of the courses taught by him and for his supportive help and advice, in general.

Also, I would like to thank Dr. Prabhu for his guidance in my master's work, and his help during my studies at Iowa State University. It was a great pleasure for me to be his student and his teaching assistant.

I am grateful to Dr. Miller for his comments on my research which enabled me to improve my work considerably. I thank him for his thoughtful concern and encouragement in my studies. Likewise, I thank Dr. McCalley and Dr. Narayanaswami for being on my committee. My collaboration with Dr. McCalley on feature subset selection was both enjoyable and fruitful, as was my interaction with Dr. Ranga Narayanaswami.

I owe a lot to my colleagues from the AI research group. First of all, I am very grateful to Rajesh Parekh for his help with various aspects of my research. Working with him on several research projects was a delight and some of the research results described in chapters 3, 4, 5 and 7 are the outgrowth of our joint work. Getting to know him personally was one of the most valuable experiences I have had at Iowa State University. Karthik Balakrishnan is also a very

good friend. I thank him for the useful discussions we had, and for his kind help in various matters. I also thank Chun-Hsien Chen and Rushi Bhatt for our fruitful research exchanges.

I give my gratitude to Raghu Havaldar for helping me to learn Java and in implementing the contract net protocol. I am also grateful to Prashant Pai for our work on agents for information retrieval. Their efforts contributed significantly to chapter 8.

I also thank many current as well as previous staff members in the computer science department office including Trish Stauble, Lynn Bremer, Melaine Eckhart, Linda Young, and Clare Polking for all that they have done for me. In addition, I am thankful to all members of the system support group including Dave Madson and Jim Schlosser for their help in using the computing facilities in the department.

Above all, I express my sincere appreciation to all my loving family– my wife Heewon, my late father and grandmother, my mother, and my sisters and brothers. Their patience, support, encouragement and sacrifices contributed greatly to my success in this endeavor.

ABSTRACT

The recent proliferation of computers and communication networks has made it possible for individuals around the world to access a wide variety of information sources through the Internet. However, effective use of these information sources requires fairly sophisticated tools or software agents for locating, classifying, selectively retrieving and extracting knowledge from data. This dissertation addresses several related research issues in the design of such intelligent agents for information retrieval and knowledge discovery from distributed data and knowledge sources.

Artificial neural networks, because of their potential for massive parallelism and fault tolerance, offer an attractive approach to the design of intelligent agents. Our work extended several single layer perceptrons and constructive neural networks of perceptrons in order to handle multi-category, real-valued patterns. In particular, we designed DistAl, a novel constructive neural network learnig algorithm based on inter-pattern distance. DistAl is significantly faster than conventional neural network algorithms and has been demonstrated to perform well on a broad variety of benchmark data-driven knowledge discovery problems. The performance of DistAl was further improved by using it in conjunction with a genetic algorithm for automated selection of features used to encode the data. DistAl was also used for data-driven refinement of incomplete or inaccurate domain knowledge. Some of these algorithms were used in a design of a multi-agent system consisting of multiple cooperating customizable intelligent mobile agents for selective information retrieval and knowledge discovery from distributed data sources.

1 INTRODUCTION

Recent advances in high throughput data acquisition technologies, digital storage technologies, computers and communications have made it possible to gather and store scientific, business, and military data in electronic form in databases and computerized information systems. In order to translate the advances in our ability to acquire and store data in increasing volumes and at increasing rates into gains in our understanding of the respective domains and new capabilities for effective decision-making, sophisticated tools are needed for information retrieval, knowledge discovery, decision-making and distributed problem-solving [Honavar *et al.*, 1998].

Several applications (e.g., military command and control, law enforcement, scientific discovery) require the use of multiple, geographically distributed, heterogeneous data and knowledge sources (e.g., sensors, satellites, intelligence reports, and so on). For instance, military commanders and intelligent analysts need to have critical information in a timely fashion to effectively perform their responsibilities. Day-to-day operations involve intelligence data gathering and analysis, situation monitoring and assessment, and looking for potentially increasing patterns in the data gathered (e.g., the relationship between troop movements and significant political developments in a region). This information can be extremely valuable for decision-making to safeguard a nation's security concerns. Therefore, it is important to have the right information at the right time at the disposal of the decision-makers instead of overwhelming them with large volumes of irrelevant data.

In this dissertation, we attempt to address several related research problems concerning the design and development of such tools, and describe some approaches to solving them.

1.1 Intelligent Agents

It is assumed here that an agent purports a *software* agent. An agent is an entity that perceives its environment and performs a set of tasks on behalf of a user with some degree of autonomy [Russell & Norvig, 1995; Honavar, 1999]. In order to do this, an agent has to embody a certain amount of *intelligence* (e.g., the ability to choose among alternative courses of action, plan, communicate, adapt to changes in the environment, and learn from experience). An agent consists of program code, a persistent internal state, and a set of attributes (e.g., movement history, authentication keys, and so on) [Russell & Norvig, 1995; Honavar, 1999].

Design of such mechanisms for intelligent agents has been the subject of study in artificial intelligence for over three decades. A broad variety of architectures for agents with differing degrees of intelligence have been proposed in the literature. These include: reactive agents which respond reactively to changes that they perceive in their environment, deliberative agents that plan and act in a goal-directed fashion, utility-driven agents that act in ways designed to maximize a suitable utility function, learning agents which modify their behavior as a function of experience, and agents that combine different modes of behavior [Russell & Norvig, 1995; Honavar, 1999].

In this dissertation, we are interested in designing agents for intelligent, selective, and context-sensitive gathering of data and its assimilation prior to analysis.

1.2 Machine Learning

Selective and context-sensitive retrieval of data requires familiarity with the user's interests. Once the data is retrieved, there is a need to extract knowledge from data. This calls for the use of techniques for automated knowledge acquisition. *Machine Learning* is probably the most practical and cost-effective approach to automated knowledge acquisition. A key objective of machine learning is to design and analyze programs that learn from experience [Langley, 1995; Mitchell, 1997; Honavar *et al.*, 1999a]. A typical machine learning system interacts with its environment, takes actions and observes their effect on the environment, and improves its performance over time.

There are several different types of machine learning approaches based on the learning mechanism used: *rote learning* (memorization); *learning through instruction*; *learning from analogy*; *deductive learning* (learning how to apply given principles); *inductive learning* (learning from examples). Among those, inductive learning approaches are the most prevalent type. Inductive learning attempts to find a succinct model to explain the examples. In other words, a *hypothesis* (represented by a proper language) that approximates (or describes) the target *concept* best is pursued in the *hypothesis space* by a well-defined procedure. In particular, we are interested in inductive learning systems for pattern classification tasks that involve classifying examples into appropriate categories. Pattern classification tasks are of interest since many real-world applications can boil down to pattern classification with appropriate transformation of representation and simplification. *Artificial neural networks* [Gallant, 1993; Haykin, 1994; Hassoun, 1995] and *decision trees* [Quinlan, 1986; Quinlan, 1993] are representative examples of inductive learning well-suited for pattern classification.

1.3 Artificial Neural Networks

An artificial neural network is a massively parallel system of simple processing units that are interconnected via trainable connection weights. Artificial neural networks have been successfully used in the design of pattern classification, function approximation, and knowledge acquisition systems. A wide variety of neural network architectures have been proposed in the literature. These differ in terms of the choice of the mathematical functions implemented by the individual neurons (processing units), the network topology (fixed or dynamic), the network architecture (number of layers and neurons), the network interconnections (connectivity among the existing neurons), the activity propagation (feedforward or recurrent), and the training methodology (one-shot or iterative) [Dayhoff, 1990].

A *perceptron* (or *Threshold Logic Unit*) [McCulloch & Pitts, 1943] is a simple neuron capable of classifying a set of patterns into two classes. The perceptron learning algorithm is a well-defined, iterative procedure for training perceptrons that guarantees correct classification for *linearly separable* sets of patterns [Rosenblatt, 1958; Minsky & Papert, 1969]. There are

a number of variants of the perceptron learning algorithm to optimize the performance (in terms of say, misclassification error) for linearly non-separable datasets while preserving the classification capability for linearly separable datasets [Gallant, 1990; Gallant, 1993; Frean, 1992; Poulard, 1995].

Constructive neural network learning algorithms [Honavar & Uhr, 1993; Gallant, 1993; Honavar *et al.*, 1999b] generate a set of neurons in a systematic, incremental way to obtain the desired accuracy on the training set. They have a significant advantage over conventional neural networks with fixed architecture in that they obviate the need for an ad hoc, a priori choice of the network topology, and possibly generate the *optimal* network with perceptron-style weight update procedures (e.g., *pocket algorithm* [Gallant, 1990; Gallant, 1993]) that are faster and simpler than the error backpropagation algorithm [Rumelhart *et al.*, 1986]. In addition, other constructive neural network learning algorithms that do not rely on the perceptron learning algorithm can be designed [Yang *et al.*, 1998b; Yang *et al.*, 1999a].

1.4 Feature Subset Selection

The performance of neural network classifiers depends critically on the choice of features used to represent the input patterns to be classified. The existence of irrelevant and redundant features can degrade the performance, cause unnecessary search effort, and bring computational overhead. Each feature also has its own measurement cost. For instance, in medical diagnosis, patterns are described using different diagnostic tests that might have different costs and risks associated with them. The *feature subset selection* task involves selecting a relevant subset of features from the entire feature set to satisfy some criteria such as improving the classification accuracy and minimizing the overall (measurement) cost of features. Constructive neural network learning algorithms can be used for the feature subset selection task (in determining the *quality* of feature subsets).

1.5 Knowledge-based Theory Refinement

Constructive neural network learning algorithms, since they obviate the need for a priori choice of the network topology and adaptively recruit neurons as needed, provide a framework for cumulative multi-task learning [Caruana, 1993; Thrun, 1995] as well as for knowledge-based theory refinement which provides a means for extending incomplete knowledge [Towell *et al.*, 1990; Fletcher & Obradović, 1993; Parekh & Honavar, 1998]. Domain specific knowledge exists in many real-world applications, and the prior knowledge (which is often incomplete) can be exploited in learning. This is because domain specific prior knowledge can easily be incorporated into the initial network configuration, and then the learning is carried out to refine the knowledge [Shavlik, 1994]. Multi-task learning can also be done in a similar way. Tasks that are learned previously can be represented by a constructive network, which can be used in learning a new task.

1.6 Mobile Agents and Multi-agent Systems

As described before, an agent is an intelligent, autonomous entity that perceives the environment and performs tasks on behalf of a user. Agents might need to move within heterogeneous networks of computers to accomplish useful tasks. Agents that can move in a computer network from host to host are called *mobile* agents. They are able to make intelligent decisions regarding their itinerary and modify it in a dynamic fashion in response to information that becomes available as they move from one host to another. Mobile agents provide a potentially efficient framework for performing computation in a distributed fashion at sites where the relevant data is available instead of expensive shipping of large volumes of data across the network. Unlike *remote procedure calls* (RPC) which require ongoing communication through a failsafe network connection from the time of initiation of the task until its completion, a major advantage of mobile agents is that ongoing interaction does not require ongoing communication [White, 1997].

The design of systems for non-trivial tasks (such as information retrieval and knowledge discovery from heterogeneous and distributed data and knowledge sources) can be simplified by

decomposing the overall task into several atomic and more manageable subtasks. Multi-agent systems are a natural consequence of this modular approach to designing complex agent-based information systems. In such multi-agent systems, satisfactory completion of the tasks depends on effective communication, negotiation, and collaboration among several independent agents that each offer a particular service associated with some cost. *KQML (Knowledge Query and Manipulation Language)* [Finin *et al.*, 1997] and *contract net protocol* [Smith, 1980] are examples of such tools that have been proposed for multi-agent coordination.

This dissertation explores several research problems related to the discussion above in the design of adaptive agents for information retrieval and data-driven knowledge discovery. The rest of the dissertation is organized as follows:

- Chapter 2

  A brief and general introduction to artificial neural networks is given. The definition, practical advantages, brief history and taxonomy of artificial neural networks are described along with their learning mechanisms.

- Chapter 3

  The perceptron learning algorithm is introduced and its limitations are explained. We also study three different variants of the perceptron learning algorithm (especially for linearly non-separable data) for both *independent* and *winner-take-all* training of neurons.

- Chapter 4

  Most of the constructive neural network learning algorithms proposed in the literature are capable of solving two-class problems with binary/bipolar inputs only. However, many real-world applications include multiple classifications and real-valued input patterns. We extend several constructive neural network learning algorithms proposed in the literature to deal with multi-category, real-valued problems [Yang *et al.*, 1996; Parekh *et al.*, 1997b; Parekh, 1998].

- Chapter 5

  The perceptron-style weight training algorithms are still iterative and time-consuming

even though they are faster than gradient-descent based learning algorithms like the error backpropagation algorithm [Rumelhart *et al.*, 1986]. We propose a new non-iterative constructive neural network algorithm, DistAl, based on the inter-pattern distances [Yang *et al.*, 1998b; Yang *et al.*, 1999a]. DistAl is significantly faster than other constructive learning algorithms and is not restricted to binary inputs or two output classes. DistAl is introduced in detail with comparative studies of its performance.

- Chapter 6

  We apply DistAl in conjunction with a genetic algorithm to the feature subset selection task to improve its performance by selecting the appropriate set of features. DistAl is used to evaluate the fitness value of each individual (i.e., possible solution). Experimental results verify the feasibility of this approach [Yang & Honavar, 1997; Yang & Honavar, 1998a; Yang & Honavar, 1998b].

- Chapter 7

  We apply DistAl to the task of theory refinement. The prior knowledge is incorporated into the initial network and then refined. Experimental results verify feasibility of this approach [Yang *et al.*, 1999b].

- Chapter 8

  We design intelligent mobile agents for retrieving relevant information. They demonstrate the feasibility and efficiency of using mobile agents for retrieving information from remote sites [Yang *et al.*, 1998d; Yang *et al.*, 1998c]. We also design a multi-agent system for information retrieval and knowledge discovery using the contract net protocol. The agents in the system negotiate and cooperate among themselves to improve the overall performance of the system. [Yang *et al.*, 1998a].

- Chapter 9

  We conclude with a summary of the research contribution of this dissertation and present some interesting directions for future research.

2 ARTIFICIAL NEURAL NETWORKS

2.1 Introduction

Artificial Neural Networks (ANN) or *connectionist networks* are massively parallel, highly interconnected, shallowly serial networks of relatively simple computing elements (or *neurons*) inspired by biological neural systems [Gallant, 1993; Honavar, 1994; Hassoun, 1995; Ripley, 1996; Mitchell, 1997; Honavar *et al.*, 1999a; Honavar *et al.*, 1999b]. The neurons are interconnected via one-way (and generally trainable) connections. Each connection has an associated *weight* that represents the strength of connection. Each neuron takes inputs from neurons it is connected to via its incoming weights and computes a function of the weighted sum of its inputs. The output is then transmitted to other neurons to which it is connected via its outgoing weights.

Learning is applied to ANN as it is happening with experience in its biological counterparts (animal brains). *Learning* plays an important role in ANN to accomplish certain tasks. Learning includes changing and finetuning some parameters in ANN with respect to well-defined criteria (e.g., minimizing the mean squared error). In general, learning involves modifying the weights in networks of a certain *a priori* chosen network architecture. It might involve modifying the network architecture as well. Learning makes use of a set of labeled examples called the *training set*. Each example is referred to as a *pattern* and is typically represented by a vector of real-valued elements. Learning in ANN roughly mimics the learning from experience that is in animal and biological brains. Typically prevalent learning in ANN involves modifying some parameters (such as connection weights) with respect to the patterns in the training set. Although ANN can adjust their behavior to accomplish tasks through learning, it should be noted that ANN are not precise models of the animal brain. It has been suggested there exist

vast differences in the models of computation in ANN and biological neural networks, and ANN are believed to have limited capabilities compared to biological neurons in the brain.

2.2 Potential Advantages of ANN

Some of the attractive features of ANN include [Gallant, 1993; Honavar, 1994; Uhr & Honavar, 1994]:

- ANN provide attractive approaches to implementing elegant learning algorithms for certain types of networks. A variety of machine learning algorithms have been developed for ANN with different characteristics (e.g., network architecture, activation function, connections, and so on) proposed to solve various problems.

- ANN are designed for massively parallel computation with each neuron operating independently and in parallel with others. Thus, ANN are well suited to the design of parallel hardware and algorithms.

- ANN have a potential for noise and fault tolerance. Since neurons work with a number of inputs, they are insensitive to noise. In other words, a larger number of correct inputs can outweigh a smaller number of incorrect inputs. Similarly, the effect of a few faulty neurons can be nullified by a larger number of properly functioning neurons.

- ANN might offer representational and computational efficiency over disjunctive normal form (DNF) expressions and decision trees. For instance, *perceptrons* (or *threshold logic units* to be described shortly) are simple and efficient in representing and computing a particular type of problems.

- ANN are similar to the networks of neurons in the brain. ANN are used as mathematical models for studying the information processing capability of the biological brain.

- ANN are shown to be Turing-equivalent as other computational models are such as *Lambda-calculus* [Church, 1941] and *post-productions* [Post, 1943]. The Turing machine is a universal model of computation in that the Turing machine can be designed to

implement any computable function [Turing, 1936]. Thus, ANN are a powerful model of computation.

ANN have been successfully applied to different types of practical problems including *pattern classification, function approximation, optimization, pattern association, clustering, vector quantization, prediction, control* and the like [Gallant, 1993; Kung, 1993; Haykin, 1994; Ripley, 1996; Zalzala & Morris, 1996; Mehrotra *et al.*, 1997; Mitchell, 1997]. In this dissertation, we focus only on pattern classification using ANN. Pattern classification involves matching a pattern with the right class it belongs to and typically makes use of a set of *training* patterns along with the corresponding class labels. ANN are trained to learn a mapping from the training patterns to their corresponding output classes. After the training, the neural network can be used to classify new formally unseen patterns.

2.3 A Brief History of ANN in Pattern Classification

In the early 1940's, McCulloch and Pitts proposed and explored a mathematical model of a biological neuron [McCulloch & Pitts, 1943]. It is variously referred to as *McCulloch-Pitts neuron, Threshold Logic Unit (TLU),* or *Perceptron.*

Hebb conceived the idea of correlating the strength of the connection weights to the activity of neurons which the weight connects [Hebb, 1949]. If both the neurons are simultaneously active (inactive), the weight connecting the neurons is increased. This *Hebbian learning* became the basis of many learning algorithms.

Rosenblatt proposed the perceptron learning algorithm to train a TLU [Rosenblatt, 1958]. It is a simple, iterative procedure to find the weight setting to compute a function implicitly specified in a set of training examples. In other words, it attempts to find a separating hyperplane that partitions the patterns space into two regions. A perceptron thus acts as a binary classifier. The perceptron learning algorithm and its variants will be described in detail in chapter 3. Perceptrons are fairly limited in their capability. Minsky and Papert pointed out the limitation of a single perceptron by demonstrating that perceptrons can solve only linearly separable problems [Minsky & Papert, 1969].

Widrow and Hoff developed a learning algorithm for perceptrons based on the *gradient descent technique* called the *Widrow-Hoff* or *delta rule* [Widrow & Hoff, 1960]. The delta rule tries to find an optimal weight setting for a given task with respect to certain criteria (e.g., minimizing the mean squared error) regardless of the linear separability of the dataset, while the perceptron learning algorithm is not guaranteed to converge to an optimal solution for non-separable datasets.

As a way around for the limitation of single layered ANN with perceptrons, several approaches and algorithms were proposed. Werbos proposed the basis of the famous *backpropagation learning algorithm* [Werbos, 1974] and several researchers developed the algorithm [Parker, 1985; Rumelhart *et al.*, 1986]. The backpropagation algorithm is a generalized version of the delta rule for networks with multiple layers, and thus is also called as *generalized delta rule*. The success of the backpropagation algorithm resuscitated the neural network community and revitalized ANN in both the theory and applications. Several variants of the backpropagation learning algorithm have appeared in the literature during the last decade.

2.4 Taxonomy

We present a taxonomy of ANN based on factors such as neuron types, network topology, and learning algorithms.

2.4.1 Neuron Types

An artificial neuron consists of a set of inputs and an output. The inputs are fed into the neuron as a vector of (typically) numeric elements. The output is computed using the inputs, the weights associated with the inputs, and the activation function associated with the neuron. The output is then propagated to adjacent neurons connected via outgoing weights.

The output is determined as follows: First, the net input (*net*) is typically computed as the weighted sum of the inputs. A neuron has an additional input connection called the *bias* or *threshold*. The input associated with the threshold is assumed to be always 1. Let $\mathbf{W} = [W_0, W_1, \cdots, W_n]$ and $\mathbf{X} = [X_0, X_1, \cdots, X_n]$ ($X_0 = 1$) be the weight and input vectors,

respectively. Then, $net = \mathbf{W} \cdot \mathbf{X} = \sum_{i=0}^{n} W_i \cdot X_i$. Second, an activation function f is applied to the net input to compute the output: $output = f(net)$. The choice of the activation function for the neurons depends on the learning algorithm and the application. Figure 2.1 depicts a neuron.

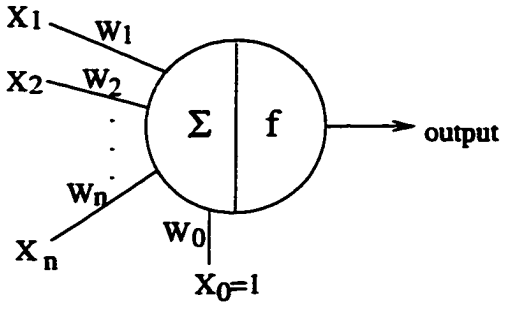


Figure 2.1 An artificial neuron.

Some of the activation functions that are widely used in the literature include:

- *Linear*

$$f(net) = net$$

- *Step*

$$f(net) = \begin{cases} a \text{ if } net > \theta \\ b \text{ if } net \leq \theta \end{cases}$$

where a and b are desired output values and θ is a threshold.

- *Sigmoid*

$$f(net) = \frac{1}{1 + e^{-net}}$$

- *Gaussian*

$$f(net) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}(\frac{net-\mu}{\sigma})^2}$$

where μ is the mean and σ is the standard deviation.

2.4.2 Network Topology

ANN consist of different sets of neurons: input, output, and possibly intermediate (or *hidden*) neurons. The input neurons receive inputs from the environment, and the output

neurons compute the final outputs of the network. Generally, the total number of input and output neurons are equal to the total number of input attributes of the training patterns and output classes in the training patterns, respectively. There might exist one or more intermediate layers between the input layer and the output layer each of which includes a set of hidden neurons. Hidden neurons are introduced for the problems that can not be solved by networks with input and output layers only. The number of hidden neurons and hidden layers depend on the learning algorithm and the task being solved. The network architecture can be either fixed a priori or determined dynamically.

ANN can be classified as *feedforward networks* or *recurrent networks*. In feedforward networks, neurons in each layer are connected only to neurons in layers above the current layer. In recurrent networks, neurons are allowed to be connected to neurons in layers below the current layer, and thus cyclic interconnections among the neurons are allowed. Figure 2.2 shows examples of networks with different topologies.

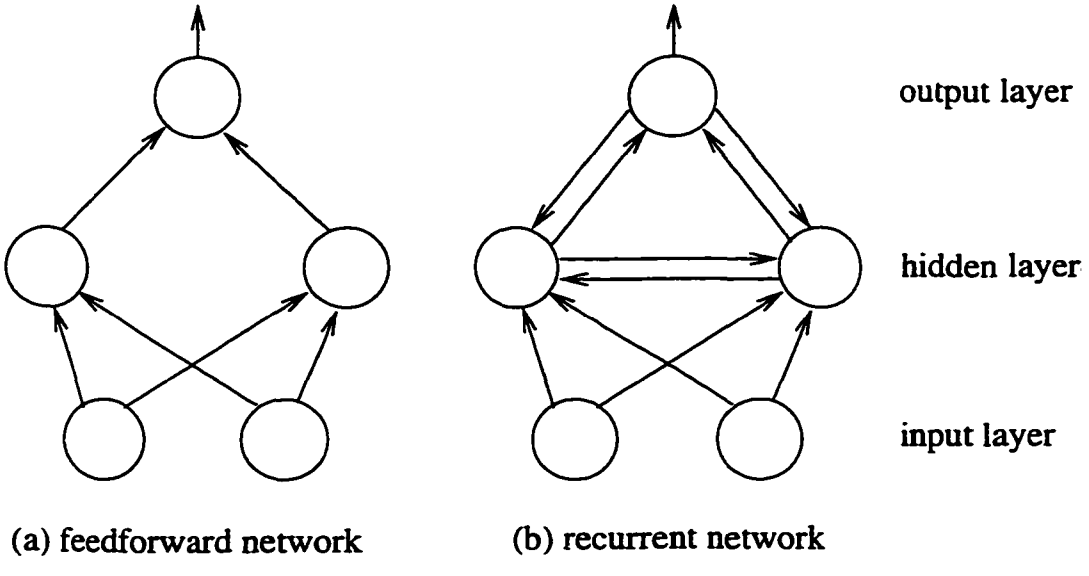


Figure 2.2 Feedforward and recurrent ANN.

2.4.3 Learning Algorithms

Learning in ANN generally involves one or both of the following approaches [Honavar, 1994]:

- *Learning as parameter modification*:

  This approach involves changing the weights in ANN with a certain a priori chosen network architecture. The number of layers, the number of hidden neurons in each hidden layer, and the connections between each neuron are defined a priori for each classification task. This is done on the basis of problem-specific knowledge (if available), or in an ad hoc fashion (requiring a process of trial and error). Finding the right architecture that has the potential for producing an optimal solution is the crucial requirement in this approach. A set of training patterns is used to finetune the weights to approximate an unknown function or to classify the patterns correctly.

  There are two different methods to update weights: First, the weights can be adjusted so as to decrease the number of misclassified patterns in the pattern space. The weight vector can be moved toward a better position geometrically for this purpose. The perceptron learning algorithm [Rosenblatt, 1958] belongs to this category. (See chapter 3 for detailed descriptions on the perceptron learning algorithm).

  Second, the weights can be adjusted by the gradient descent technique using a well-defined objective function. The *backpropagation algorithm* [Parker, 1985; Rumelhart *et al.*, 1986] is an example of this method. There are two phases in the backpropagation algorithm: forward and backward. In the forward phase, the pattern is presented, outputs are produced in intermediate neurons, and final outputs are determined in the output layer. Then, the objective function (typically the mean squared error) is computed. In the backward phase, the error is propagated backward and the weights (and other modifiable parameters) are adjusted in the direction of negative gradient of the objective function. This process is repeated iteratively until the objective function reaches a (local) minimum. See [Dayhoff, 1990; Gallant, 1993; Mitchell, 1997] for detailed explanation on backpropagation algorithm and the derivation of weight update rule. The backpropga-

tion algorithm is perhaps the most popular and practical algorithm and has been used in many applications. However, it has the following drawbacks as well: slow learning and local minima. The algorithm requires expensive computations for error propagation and iterative weight updates. The algorithm typically requires smaller learning rates in order to converge to a solution, which entails a large number of iterations. Even though several approaches are proposed to speed up the learning (e.g., use of momentum constant), backpropagation algorithm has a significant disadvantage in terms of speed. In addition, the gradient descent method does not guarantee the global minimum. In the search space with multiple minima the algorithm is susceptible to local minima.

- *Learning by changing network architecture*:

  This approach is called a *constructive* or *generative* neural network learning [Honavar, 1990; Honavar & Uhr, 1993]. It attempts to find an appropriate network architecture for a given task as well as proper weight settings within the network architecture. This is motivated by the fact that there is no guarantee of discovering the solution weights within the search space defined by the network architecture and the search algorithm chosen a priori. The topology of the target network is determined dynamically by introducing new neurons, layers, and connections in a controlled fashion. In some cases, pruning mechanisms that discard redundant neurons and connections are used in conjunction with the network construction mechanisms [Reed, 1993; Parekh *et al.*, 1997c]. The problem of determining the network architecture a priori in the fixed architecture approach no longer exists in this case, and the chance of discovering a near minimal network increases. Finding parsimonious or compact network topologies is of interest because they provide a potential for simpler hardware design, easier interpretation of the network, better matching of the intrinsic complexity of the given task, superior generalization capability, and so on. While networks with a fixed architecture do not even guarantee the existence of solutions in the space defined by the architecture, constructive learning algorithms attempt to find a suitable architecture that contains a solution.

  A variety of constructive neural network learning algorithms have been proposed in

the literature. We will explore several constructive neural network learning algorithms that use perceptrons. We will also propose a new constructive neural network learning algorithm based on the inter-pattern distances of patterns. These will be described in detail in chapter 4 and chapter 5, respectively.

3 PERCEPTRON LEARNING ALGORITHMS

3.1 Introduction

A perceptron (or TLU) is a simple mathematical model motivated by the McCulloch and Pitts model of the biological neuron [McCulloch & Pitts, 1943]. A perceptron implements the step function and can be trained to classify a set of patterns into two classes. Consider a pattern set defined over the N-dimensional Euclidean space. The output O^p of a perceptron with a weight vector $\mathbf{W} = (W_0, W_1, \cdots, W_N)$ for an input pattern $\mathbf{X}^p = (X_0^p, X_1^p, \cdots, X_N^p)$ is computed as follows:

$$O^p = \begin{cases} 1 \text{ if } \mathbf{W} \cdot \mathbf{X}^p > 0 \\ -1 \text{ otherwise} \end{cases}$$

The pattern classification properties of a perceptron (and networks of perceptrons) are better understood in geometrical terms [Nilsson, 1965; Chen *et al.*, 1995]. A perceptron implements a $(N-1)$-dimensional hyperplane given by $\mathbf{W} \cdot \mathbf{X}^p = 0$. The hyperplane partitions the N-dimensional pattern space (defined by the coordinates $X_1, \cdots, X_N$) into two regions.

3.1.1 Limitation of Perceptrons

A perceptron is only capable of correctly classifying patterns that are *linearly separable*. A set of patterns $S = S^+ \cup S^-$ where $S^+ = \{(\mathbf{X}^p, C^p) \,|\, C^p = 1\}$ and $S^- = \{(\mathbf{X}^p, C^p) \,|\, C^p = -1\}$ (C^p is the desired output for the input pattern $\mathbf{X}^p$) is said to be linearly separable if there exists a weight vector $\hat{\mathbf{W}}$ such that $\forall \mathbf{X}^p \in S^+, \hat{\mathbf{W}} \cdot \mathbf{X}^p > 0$ and $\forall \mathbf{X}^q \in S^-, \hat{\mathbf{W}} \cdot \mathbf{X}^q \leq 0$. For instance, consider the **AND** pattern set $S = \{[(-1\ -1),\ -1], [(-1\ 1),\ -1], [(1\ -1),\ -1], [(1\ 1),\ 1]\}$. The **AND** pattern set is linearly separable since there exist lots of hyperplanes separating the

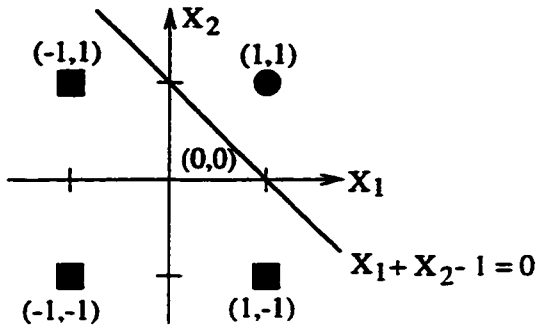


Figure 3.1 **AND** pattern set.

patterns into two classes including the one with $\mathbf{W} = [-1\ 1\ 1]$ as shown in Figure 3.1. Note that the first component of the weight vector is the threshold (W_0).

On the contrary, there are patterns sets for which there does not exist a hyperplane to separate the pattern sets into two regions. In other words, perceptrons alone are not sufficient to implement arbitrarily complex decision regions that may be necessary to deal with linearly non-separable training sets. For instance, consider the **XOR** pattern set $S = \{[(-1\ -1),\ 1], [(-1\ 1),\ -1], [(1\ -1),\ -1], [(1\ 1),\ 1]\}$ in Figure 3.2. The patterns can never be separated into corresponding classes by any hyperplanes. A perceptron has this limited capability of handling linearly non-separable training sets.

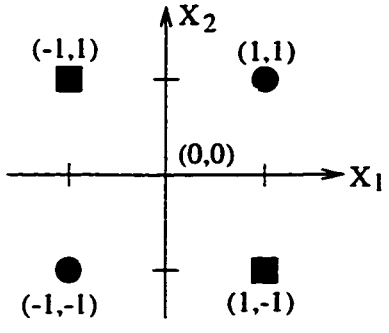


Figure 3.2 **XOR** pattern set.

3.1.2 Learning Algorithm

The following learning algorithm is proposed for perceptrons [Rosenblatt, 1958; Nilsson, 1965; Minsky & Papert, 1969]:

$$\mathbf{W} - \mathbf{W} + \eta(D^p - O^p)\mathbf{X}^p$$

where $\mathbf{W}$ is the weight vector, $\mathbf{X}^p$ is the pattern vector, D^p is the desired output and O^p is the computed output for $\mathbf{X}^p$, and $\eta > 0$ is the learning rate, respectively. The perceptron algorithm updates weights iteratively by adding (or subtracting) a fraction of the misclassified pattern to the current weight vector in a bid to correctly classify as many patterns as possible as learning proceeds. The perceptron weight update rule is guaranteed to converge to a weight vector of the separating hyperplane in the pattern space if one exists. However, if the dataset is not linearly separable, the behavior of the perceptron learning algorithm is unpredictable in the sense that the classification accuracy might fluctuate from iteration to iteration.

A single perceptron is capable of classifying a set of linearly separable patterns into two classes. However, a number of practical applications involve a *multi-category* classification where there are $M(M > 2)$ output classes. An extension of the simple perceptron model to multiple output categories is rather straightforward with one TLU being allocated per output category. Assuming that the training patterns belong to M output categories, the M TLUs can be trained either *independently* or as a *winner-take-all (WTA)* group.

The independent training is similar to 2-category classification which involves just a single TLU. For M categories, the TLU i is trained with S_+ being the set of patterns belonging to class i and S_- being the set of patterns belonging to all the other classes. However, the independent training does not consider the interrelationships among the different pattern classes.

The WTA output strategy takes into account the fact that each pattern can belong to only one output class. Here the weight updates are geared toward pushing the target TLU to have the highest net input among the group of M TLUs. Let $\mathbf{D}^p$ and $\mathbf{O}^p$ represent the desired and obtained output vectors in response to input pattern $\mathbf{X}^p$ respectively. The weight vectors of the M TLUs are $\mathbf{W}_1, \mathbf{W}_2, \cdots, \mathbf{W}_M$ respectively and $\mathbf{O}^p = [O_1^p, O_2^p, \cdots, O_M^p]$. $\mathbf{O}^p$ is computed

as follows: If $\exists j \in 1, \cdots, M$ such that $\mathbf{W}_j \cdot \mathbf{X}^p > \mathbf{W}_i \cdot \mathbf{X}^p \; \forall i \neq j, i = 1, \cdots, M$ then $O_j^p = 1$ and $O_i^p = -1$, $i \neq j$. If $\exists j_1, j_2, \cdots, j_k \in 1, \cdots, M$ such that $\mathbf{W}_{j_1} \cdot \mathbf{X}^p = \mathbf{W}_{j_2} \cdot \mathbf{X}^p = \cdots = \mathbf{W}_{j_k} \cdot \mathbf{X}^p$ and $\mathbf{W}_{j_1} \cdot \mathbf{X}^p > \mathbf{W}_i \cdot \mathbf{X}^p \; \forall i \notin \{j_1, j_2, \cdots, j_k\}$ then $O_j^p = -1, \; \forall j \in 1, \cdots, M$. The weights are then modified according to the perceptron weight update rule as independent training: $\mathbf{W}_j \leftarrow \mathbf{W}_j + \eta(D_j^p - O_j^p)\mathbf{X}^p \; \forall j \in 1, \cdots, M$.

The WTA training offers a significant advantage over independent training in that pattern classes that are only pairwise separable from each other can be correctly classified using WTA while in independent training only pattern classes that are independently separable from each other can be correctly classified. Figure 3.3 depicts a set of patterns with three classes that can be classified by WTA training (i.e., pairwise separable) but not by independent training.

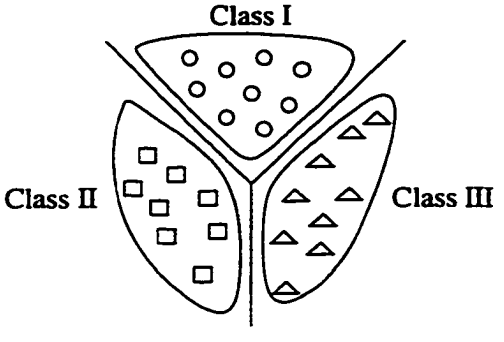


Figure 3.3 An example of pairwise separable pattern set.

Some advantages of the perceptron family of learning algorithms include the existence of well-known convergence results [Rosenblatt, 1958; Nilsson, 1965; Minsky & Papert, 1969], and substantially faster learning as compared to typical gradient-based error minimization strategies. However, they also have the following limitations:

- They behave poorly on datasets that are not linearly separable - i.e., the classification accuracy on the training set can fluctuate considerably from iteration to iteration [Gallant, 1993].
- They alone are not sufficient to implement arbitrarily complex decision regions that may be necessary to deal with training sets that are not linearly separable.

The focus of this chapter is on variants of perceptron algorithms that address these two limitations. These variants, while preserving the convergence properties of the perceptron algorithm on linearly separable data, attempt to find *near-optimal* weights (so as to correctly classify as large a fraction of the training set as possible) when the training dataset is not linearly separable. One approach to overcome the second limitation is to use *generative* or *constructive* learning algorithms [Honavar & Uhr, 1993; Gallant, 1993; Parekh, 1998; Honavar *et al.*, 1999b]. Such constructive algorithms rely on the addition of typically one (but in some cases, a few) neurons at a time to build a multi-layer perceptron that correctly classifies a given training set. Each added neuron is trained using an appropriate weight modification algorithm. Since constructive learning algorithms are designed to deal with non-linearly separable datasets, the behavior of the weight modification routine on such data is critical to their performance. (See chapter 4 for constructive learning algorithms). A perceptron learning algorithm has its own *inductive bias*. In other words, the process of determining the weights is different in each algorithm. The bias makes a particular algorithm more suitable to a particular problem and yields a better performance than other algorithms. It is against this background that we approach our study of the performance of variants of perceptron algorithms on non-linearly separable datasets.

A number of variants of the perceptron learning algorithms have been proposed by many researchers [Krauth & Mézard, 1987; Anlauf & Biehl, 1990; Gallant, 1990; Frean, 1992; Gallant, 1993; Poulard, 1995; Raffin & Gordon, 1995]. Recently, the necessary and sufficient conditions for a pattern set to be linearly non-separable were given in [Siu *et al.*, 1995]. It was also shown that the problem of identifying a largest linearly separable subset of a pattern set is NP-complete. Therefore, the primary motivation for the variants is to find *near-optimal* weights (so as to correctly classify as large a fraction of the training set as possible) when the training dataset is not linearly separable, while preserving the convergence properties of the perceptron algorithm on linearly separable data. This chapter considers the following three algorithms among many approaches: *pocket algorithm* [Gallant, 1990; Gallant, 1993], *thermal perceptron* [Frean, 1992] and *barycentric correction procedure* [Poulard, 1995]. The performance of the

algorithms is compared using a variety of both real-world and toy datasets. A majority of these datasets are linearly non-separable. The three algorithms give 100% training accuracy on datasets that are linearly separable and attempt to classify as large a subset of the training set as possible in the case of non-linearly separable datasets.

The rest of this chapter is organized as follows: Section 3.2 introduces the three variants of the perceptron learning algorithm, provides the pseudo code and analyzes the time and space complexity of each algorithm. Section 3.3 describes the datasets used in experiments. Section 3.4 presents the results of the comparative experiments of three algorithms. Section 3.5 concludes with a summary and discussion of future research.

3.2 Description of Three Variants of Perceptron Learning Algorithm

3.2.1 Pocket Algorithm

The perceptron algorithm updates weights iteratively by adding (or subtracting) a fraction of the misclassified pattern to the current weight vector in a bid to correctly classify as many patterns as possible as learning proceeds.

The key idea behind the *pocket algorithm* [Gallant, 1990; Gallant, 1993] which is explicitly designed to improve the behavior of the perceptron algorithm on non-linearly separable data is to maintain an additional weight vector $\mathbf{W}_{pocket}$ in addition to current $\mathbf{W}$. $\mathbf{W}_{pocket}$ stores the best weight setting encountered during training. A further refinement on this idea, called the *ratchet* modification (RP) [Gallant, 1993], is to ensure that replacement of $\mathbf{W}_{pocket}$ by $\mathbf{W}$ is performed only if $\mathbf{W}$ correctly classifies a greater fraction of the training set than $\mathbf{W}_{pocket}$. The pocket convergence theorem [Gallant, 1993] guarantees that the RP will find an optimal weight setting given enough training time.

In the following discussion on multi-category algorithms, $\mathbf{W}$ and $\mathbf{W}_{pocket}$ denote the entire set of weight vectors of M output neurons in the network. $\mathbf{W}_j$ represents a specific weight vector of neuron j. The pseudo code for pocket algorithm with ratchet modification is depicted in Figure 3.4.

1. Initialize $\mathbf{W}$ (can be initialized to 0 or small random values);
2. for k := 1 to (# of epochs) do
3. Select a training example $(\mathbf{X}^p, \mathbf{D}^p)$ at random;
4. Compute the output vector $(\mathbf{O}^p)$;
5. if $(\mathbf{O}^p = \mathbf{D}^p)$ then // correct classification
6. if (run of correct classification with $\mathbf{W}$ is longer than that with $\mathbf{W}_{pocket}$) then
7. if ($\mathbf{W}$ correctly classifies more training examples than $\mathbf{W}_{pocket}$) then
8. Replace $\mathbf{W}_{pocket}$ by $\mathbf{W}$ and adjust the length of correct run;

endif

endif

else // incorrect classification

9. Update weight vectors: $\mathbf{W}_j \leftarrow \mathbf{W}_j + \eta \cdot (D_j^p - O_j^p)\mathbf{X}^p, \forall j \in 1, \cdots, M$

endif

Figure 3.4 Pocket algorithm with ratchet modification.

3.2.1.1 Time Complexity

Let N_{epoch} be the total number of epochs for which the algorithm is trained. Let N_{in} and N_{out} be the number of input and output neurons respectively. Let $N_{pattern}$ be the number of training patterns.

Step 1,4 and 8 take $\mathcal{O}(N_{in} \cdot N_{out})$. Step 3 and 6 take $\mathcal{O}(1)$. Step 5 takes $\mathcal{O}(N_{out})$. Step 7 takes $\mathcal{O}(N_{pattern} \cdot N_{in} \cdot N_{out})$. Step 9 takes $\mathcal{O}(N_{in} \cdot N_{out})$. Thus, the total time complexity (at step 2) is $\mathcal{O}(N_{epoch} \cdot N_{pattern} \cdot N_{in} \cdot N_{out})$.

3.2.1.2 Space Complexity

The space requirement for input patterns and their targets is $\mathcal{O}(N_{pattern} \cdot N_{in} \cdot N_{out})$. and for $\mathbf{W}_{pocket}$ and $\mathbf{W}$ is $\mathcal{O}(N_{in} \cdot N_{out})$. Thus, the overall space complexity is $\mathcal{O}(N_{pattern} \cdot N_{in} \cdot N_{out})$.

3.2.2 Thermal Perceptron Algorithm

The rationale behind the thermal perceptron algorithm (TP) [Frean, 1992] is to control the weight updates to avoid drastic changes for outliers as learning progresses. The fact that the weight update rule of the standard perceptron algorithm for misclassifications is the same irrespective of the magnitude of the error can cause severe fluctuations in the classification rate for non-separable datasets. A damping factor is introduced in the weight update equation to stabilize learning: $\mathbf{W}_j \leftarrow \mathbf{W}_j + \eta \frac{1}{T}(D_j^p - O_j^p)\mathbf{X}^p e^{-|\phi|/T}$ where ϕ is the net input for the output neuron and T is the temperature. The temperature T is set to an initial value T_0 at the start of learning and gradually annealed to 0 as the training progresses. Since the exponent effectively decays the learning rate, the probability of undoing previous work is decreased as training progresses. In effect, the algorithm behaves like the perceptron algorithm at the start and avoids any large weight changes at the end of training. Note that the performance of this algorithm is heavily dependent on the initial temperature. This difficulty can be overcome to a significant extent if at the end of each epoch the initial temperature T_0 is set to the average net input over that particular epoch [Burgess, 1994]. The TP can be directly applied to multi-category classification problems using WTA computation as in RP. However, it is reasonable to account

for the interactions between output neurons in the computation of ϕ. In other words, the difference of net inputs between target output neuron and the neuron with highest net input (step 8 of the pseudo-code in Section 3.2.2) is used as ϕ in the weight update formula. (In fact, direct extension of TP to WTA groups was found to perform poorly and thus the above heuristic was incorporated). The pseudo code for thermal perceptron algorithm is depicted in Figure 3.5.

1. Initialize $\mathbf{W}$ (can be initialized to 0 or small random values);
2. Set initial temperature $T_0 = 1$; $\gamma = 1$; $T = \gamma T_0$;
3. for k := 1 to (# of epochs) do
4. for l := 1 to (# of patterns) do
5. Select a training example $(\mathbf{X}^p, \mathbf{D}^p)$ at random;
6. Compute the output vector $(\mathbf{O}^p)$;
7. if $(\mathbf{O}^p \neq \mathbf{D}^p)$ then // incorrect classification
8. Update weight vectors:
   - Independent:
     $$\mathbf{W}_j - \mathbf{W}_j + \eta \frac{1}{T} \cdot (D_j^p - O_j^p)\mathbf{X}^p e^{-|\mathbf{W}_j \cdot \mathbf{X}^p|/T}, \forall j \in 1, \cdots, M$$
   - WTA (neuron i has the highest net input):
     $$\mathbf{W}_j - \mathbf{W}_j + \eta \frac{1}{T} \cdot (D_j^p - O_j^p)\mathbf{X}^p e^{-|\mathbf{W}_i \cdot \mathbf{X}^p - \mathbf{W}_j \cdot \mathbf{X}^p|/T}, \forall j \in 1, \cdots, M$$

   endif
9. Compute the average net input (ϕ_{avg}) over all output neurons:
10. $\gamma = \gamma - (1/\#$ of Epoch$)$: $T_0 = (2T_0 + 2\phi_{avg})/3$; $T = \gamma T_0$;

Figure 3.5 Thermal perceptron algorithm.

3.2.2.1 Time Complexity

We use the same notation as described in RP. Step 1 and 6 take $\mathcal{O}(N_{in} \cdot N_{out})$. Step 2, 5, 9 and 10 take $\mathcal{O}(1)$. Step 7 takes $\mathcal{O}(N_{out})$. Step 8 takes $\mathcal{O}(N_{in} \cdot N_{out})$. Thus, step 4 takes $\mathcal{O}(N_{pattern} \cdot N_{in} \cdot N_{out})$. Therefore, the total time complexity (at step 3) is $\mathcal{O}(N_{epoch} \cdot N_{pattern} \cdot N_{in} \cdot N_{out})$.

Note that TP has a merit of not requiring the ratchet test (i.e., computing the overall training accuracy), but it depends on expensive exponent calculations and floating point arithmetic.

3.2.2.2 Space Complexity

As in RP, input patterns and their targets need $\mathcal{O}(N_{pattern} \cdot N_{in} \cdot N_{out})$, and $\mathbf{W}$ requires $\mathcal{O}(N_{in} \cdot N_{out})$. Thus, the overall space complexity is $\mathcal{O}(N_{pattern} \cdot N_{in} \cdot N_{out})$.

3.2.3 Barycentric Correction Procedure

The barycentric correction procedure (BCP) [Poulard, 1995] is an efficient algorithm for training single layer neurons. It is based on the geometric properties of the training patterns and provides a framework for rapidly determining a stable weight setting that correctly classifies as large a subset of the training patterns as possible.

The BCP algorithm for two-category classification involves iteratively computing the barycenters for each of the two classes and the threshold in a bid to minimize the number of misclassifications. The BCP features separate methods for computing the weights and the threshold of the TLU being trained. Let N_1 and N_0 be the number of patterns in $\mathcal{S}_+$ and $\mathcal{S}_-$ respectively. The barycenters $\mathbf{b}_1$ and $\mathbf{b}_0$ represent the weighted averages of the patterns in $\mathcal{S}_+$ and $\mathcal{S}_-$ respectively with $\alpha = (\alpha_1, \cdots, \alpha_{N_1})$ and $\mu = (\mu_1, \cdots, \mu_{N_0})$ representing the weighting coefficients for patterns belonging to $\mathcal{S}_+$ and $\mathcal{S}_-$ respectively. The weight vector $\mathbf{W} = (w_1, \cdots, w_N)$ is determined as $\mathbf{W} = \mathbf{b}_1 - \mathbf{b}_0$. The threshold θ is then chosen to optimize classification accuracy. The sets $\nu_1 = \{-\mathbf{W} \cdot \mathbf{X}^p | \mathbf{X}^p \in \mathcal{S}_+\}$ and $\nu_0 = \{-\mathbf{W} \cdot \mathbf{X}^p | \mathbf{X}^p \in \mathcal{S}_-\}$ representing the projections of the individual patterns on the weight vector $\mathbf{W}$ are first computed. If $\max(\nu_1) < \min(\nu_0)$ it is clear that the projections of the patterns belonging to the two classes do not overlap and hence

the patterns $\mathcal{S}_+$ and $\mathcal{S}_-$ are linearly separable. θ is set to $\frac{\max(\nu_1)+\min(\nu_0)}{2}$ and the separating hyperplane is given by $\mathcal{H} = (\theta, \mathbf{W})$. If however, $\max(\nu_1) \geq \min(\nu_0)$ then the pattern set is not linearly separable and θ is selected randomly from the interval $[-\mathbf{W} \cdot \mathbf{b}_1 \; .. \; -\mathbf{W} \cdot \mathbf{b}_0]$. Like RP, BCP maintains a pocket hyperplane $\mathcal{H}_{pocket} = (\theta_{pocket}, \mathbf{W}_{pocket})$ capturing the threshold and weights encountered during training that together give minimum classification error. For each training epoch i the candidate pocket hyperplane is denoted as $\mathcal{H}^i_{pocket} = (\theta^i_{pocket}, \mathbf{W})$. θ^i_{pocket} is selected from a pool of values representing the overlapping region of patterns belonging to both classes in the set of projections $\nu = \nu_1 \cup \nu_0$. The separation (or *gap*) of the patterns from the hyperplane $\mathcal{H}^i_{pocket}$ is computed as the sum of the distances from $\mathcal{H}^i_{pocket}$ of the closest patterns on either side. Finally, $\mathcal{H}^i_{pocket}$ replaces the current pocket hyperplane ($\mathcal{H}_{pocket}$) if the number of misclassifications is less than the number of misclassifications of $\mathcal{H}_{pocket}$ or if the number of misclassifications is the same and the gap is greater than the gap of $\mathcal{H}_{pocket}$. To end one epoch, the weighting coefficients of the patterns that are still misclassified are boosted up by a positive weighting modification. Intuitively, this causes the misclassified patterns of the two classes to be weighted more heavily in the computation of the barycenters. Training is performed for a prespecified number of epochs at the end of which the best weights represented by the pocket hyperplane are returned.

The multicategory extension of the BCP is implemented as a sequence of M calls to the two-category BCP procedure once for each of the M pattern classes. The training set for output neuron j is constructed by assigning target output 1 to patterns of class j and output -1 to all other patterns.

The extension of the BCP to WTA groups involves treating the barycenters for each class of patterns as the weights for the corresponding neuron. The thresholds for the neurons are then determined by minimizing the *loss* due to misclassification. The loss-minimization algorithm [Hrycej, 1992] can be adapted for this purpose. Of course, the loss minimization algorithm can be used by itself as a weight training rule for the weights of the TLU. However, the convergence speed of this process which is based on loss minimization by gradient decent is excruciatingly slow for the complex loss surface when it is used to train the weights and the

threshold. Thus we use the loss minimization routine to compute just the thresholds in this case. Given the set of weights, the loss can be defined as the sum squared error incurred in classifying each pattern. Suppose the TLU numbered i with weight vector $\mathbf{W}_i$ (remember that in the case of the BCP the threshold θ_i is computed separately from the weight vector) produces the highest activation among all the neurons for pattern $\mathbf{X}_p$. Suppose j is the correct classification for pattern $\mathbf{X}^p$. Then the cumulative loss for the training patterns is given by $Q = \sum_p (n_i^p - n_j^p)^2$ where $n_i^p = \mathbf{X}^p \cdot \mathbf{W}_i$ is the activation of neuron i in response to pattern $\mathbf{X}^p$. It can be shown that the cumulative loss function is a convex differentiable function of the modifiable thresholds, and consequently, has a unique minimum [Shynk, 1990; Hrycej, 1992]. The thresholds that correspond to the minimum value of Q are found by gradient descent. It is easy to prove that such a learning rule is guaranteed to find a set of separating thresholds if the training set is linearly separable. Even if the training set is not linearly separable, this method guarantees to find the thresholds that minimize the cumulative loss and hence maximize the number of correctly classified patterns. However, the quality of the solution is a function of the distribution of patterns in the pattern space. Because the purpose of introducing loss minimization algorithm here is to find the optimal thresholds to minimize the misclassification and use the routine as an inner loop of WTA BCP, we provide a limited number of iterations to perform the gradient descent instead of allowing indefinite training time. Finally, to obviate the oscillation of cumulative loss due to large learning rate, we dynamically decrease the learning rate η if the cumulative loss diverges during the gradient descent.

The pseudo code in Figure 3.6 is the independent BCP for two class problems. For multi-category problems, this procedure is run several times (once for each category) as described before.

The pseudo code for WTA barycentric correction procedure is depicted in Figure 3.7. Let α denote a collection of weighting coefficients α_j. Let $\mathbf{W}$ (and $\mathbf{W}_{pocket}$) denote a collection of weight vectors $\mathbf{W}_j$ (and pocket vectors $\mathbf{W}_{pocket}$) for each class j. Let $\Theta = [\theta_1, \cdots, \theta_M]$ be the thresholds of the M neurons and Θ_{pocket} be the pocket thresholds.

1. Initialize α and μ to values in the range $[1, a]$;
2. for k := 1 to (# of epochs) do
3. Compute $\mathbf{b_1}$ and $\mathbf{b_0}$;
4. Set $\mathbf{W} = \mathbf{b_1} - \mathbf{b_0}$;
5. Compute ν_1 and ν_0;
6. if $(\max \nu_1 < \min \nu_0)$ then begin;
7. Set $\theta = \frac{\max \nu_1 + \min \nu_0}{2}$;
8. Return $\mathcal{H} = [\theta, \mathbf{W}]$ and stop;

else

9. Pick θ randomly from $[-\mathbf{W} \cdot \mathbf{b_1} \; .. \; -\mathbf{W} \cdot \mathbf{b_0}]$;
10. Compute the candidate θ^i_{pocket} and the gap of $\mathcal{H}^i_{pocket}$;
11. Replace the pocket hyperplane $\mathcal{H}_{pocket}$ by $\mathcal{H}^i_{pocket}$ if $\mathcal{H}^i_{pocket}$ correctly classifies more training examples than $\mathcal{H}_{pocket}$ or has same classification accuracy and a larger gap;
12. Update α and μ;

endif

Figure 3.6 Barycentric correction procedure (independent).

1. Initialize α;
2. for k := 1 to (# of epochs) do
3. for j := 1 to M do
4. Compute $\mathbf{b}_j$;
5. Set $\mathbf{W}_j = \mathbf{b}_j$;
6. Determine Θ by loss minimization;
7. if (all patterns are correctly classified by $\mathcal{H} = [\Theta, \mathbf{W}]$) then
8. return $\mathcal{H}$ and stop;
9. else if ($\mathcal{H}$ results in fewer errors compared to $\mathcal{H}_{pocket} = [\Theta_{pocket}, \mathbf{W}_{pocket}]$) then
10. $\mathcal{H} \leftarrow \mathcal{H}_{pocket}$;

endif

11. Update α;

Figure 3.7 Barycentric correction procedure (WTA).

3.2.3.1 Time Complexity

Using the same notation as in RP we analyze the time complexity as:

1. Independent BCP

   Step 1, 3, 6 and 12 take $\mathcal{O}(N_{pattern})$. Step 4 takes $\mathcal{O}(N_{in})$. Step 5 and 11 take $\mathcal{O}(N_{pattern} \cdot N_{in})$. Step 7, 8 and 9 take $\mathcal{O}(1)$. Step 10 takes $\mathcal{O}(N_{pattern} \lg N_{pattern})$. Thus, the total time complexity (at step 2 with multicategory) is $\mathcal{O}(\max[N_{out} \cdot N_{epoch} \cdot N_{pattern} \lg N_{pattern}, N_{out} \cdot N_{epoch} \cdot N_{pattern} \cdot N_{in}])$. (For multicategory problems, the algorithm should be run for each output.)

2. WTA BCP

   Step 1, 4 and 11 take $\mathcal{O}(N_{pattern})$. Step 5 takes $\mathcal{O}(N_{in})$, and therefore step 3 takes $\mathcal{O}(\max[N_{out} \cdot N_{in}, N_{out} \cdot N_{pattern}])$. Step 6 takes $\mathcal{O}(N_{innerepoch} \cdot N_{pattern} \cdot N_{in} \cdot N_{out})$ (*innerepoch* is needed for loss minimization). Step 7 takes $\mathcal{O}(N_{pattern} \cdot N_{in})$. Step 8 and 9 take $\mathcal{O}(1)$. Step 10 takes $\mathcal{O}(N_{in} \cdot N_{out})$. Thus, the total time complexity (at step 2) is $\mathcal{O}(N_{epoch} \cdot N_{pattern} \cdot N_{in} \cdot N_{out})$.

3.2.3.2 Space Complexity

As in RP, the space requirement for the input patterns and their targets is $\mathcal{O}(N_{pattern} \cdot N_{in} \cdot N_{out})$. $\mathbf{W}$, $\mathbf{W}_{pocket}$ and $\mathbf{b}$ require $\mathcal{O}(N_{in} \cdot N_{out})$, and Θ and Θ_{pocket} require $\mathcal{O}(N_{out})$. Thus, the overall space complexity is $\mathcal{O}(N_{pattern} \cdot N_{in} \cdot N_{out})$.

3.3 Datasets

In order to conduct a thorough and systematic comparison between the three algorithms, a wide range of datasets was chosen based on a set of carefully chosen criteria which involved:

- *Attribute Type*: binary/bipolar, integer and real valued attributes.
- *Number of Output Categories*: two classes or multiple output classes.
- *Linear separability*: separable and non-separable sets of training patterns.

The real-world datasets used are available at UC Irvine's Machine Learning repository [Murphy & Aha, 1994]. Table 3.1 summarizes the characteristics of the datasets selected for our experiments. Train and Test are the number of patterns in the training and test sets, respectively. Attribute is the number of input attributes. Class is the number of output classes.

Table 3.1 Datasets used in the experiments.

Dataset	Train	Test	Attribute	Attribute Type	Class
balance	416	209	4	real	3
concentric (two concentric circles)	1666	834	2	real	2
glass	142	72	9	real	6
ionosphere	234	117	34	real	2
liver	230	115	6	real	2
p7 (7-bit parity)	128	0	7	bipolar	2
pima	512	256	8	real	2
r5 (5-bit random)	32	0	5	bipolar	3
sep (separable data)	200	100	4	real	2
soybean	33	14	35	integer	4
wdbc	380	189	30	real	2
wine	120	58	13	real	3
WTA-sep (separable data in WTA sense)	44	21	2	real	5

3.4 Experiments and Results

Several experiments were conducted to make a fair comparison between the three perceptron algorithms (BCP, RP and TP) in terms of classification accuracy, training time, and learning curve. Also, a constructive learning algorithm was chosen to study the inductive bias of the perceptron algorithms in the constructive learning algorithm.

3.4.1 Classification Accuracy

To compare the classification accuracy, sufficient learning time (in terms of the number of *epochs*) was allowed to each algorithm. An epoch indicates a single random pattern presentation in RP, l randomly drawn patterns from the training set in TP (where l is the size of the training set), and a presentation of the entire set of training patterns in BCP. A run of RP was terminated upon attaining 100% accuracy on the training data or when the pocket weights

Table 3.2 Classification accuracy (independent).

	RP		TP		BCP	
Dataset	*train*	*test*	*train*	*test*	*train*	*test*
balance	88.5 ± 0.5	85.6 ± 0.8	84.3 ± 2.8	82.2 ± 1.4	87.7 ± 0.5	82.1 ± 1.1
concentric	62.7 ± 0.0	64.0 ± 0.0	62.7 ± 0.0	64.0 ± 0.0	55.9 ± 0.0	53.1 ± 0.1
glass	46.8 ± 1.6	41.8 ± 4.5	39.1 ± 2.1	28.9 ± 2.9	54.4 ± 2.0	40.7 ± 4.1
ionosphere	91.9 ± 0.9	95.0 ± 1.7	95.4 ± 0.5	91.6 ± 2.3	91.9 ± 0.2	95.1 ± 0.9
liver	70.7 ± 1.0	70.8 ± 3.0	71.7 ± 0.9	72.1 ± 2.6	72.0 ± 0.3	70.2 ± 1.7
p7	61.0 ± 4.5	–	65.2 ± 1.6	–	63.6 ± 3.8	–
pima	68.9 ± 1.0	68.7 ± 2.0	72.0 ± 0.8	71.0 ± 1.4	74.6 ± 0.4	77.8 ± 0.7
r5	56.9 ± 3.4	–	58.6 ± 4.9	–	57.6 ± 3.8	–
sep	100 ± 0.0	99.8 ± 0.5	100 ± 0.0	99.8 ± 0.5	100 ± 0.0	99.8 ± 0.4
soybean	100 ± 0.0	92.3 ± 7.4	100 ± 0.0	94.3 ± 4.6	100 ± 0.0	80.9 ± 4.0
wdbc	92.3 ± 0.3	91.4 ± 4.1	92.7 ± 0.2	90.8 ± 1.4	90.3 ± 0.0	91.6 ± 0.2
wine	71.8 ± 6.3	75.9 ± 10.3	77.1 ± 1.5	85.4 ± 2.2	73.9 ± 3.9	83.2 ± 3.3
WTA-sep	71.1 ± 1.4	78.7 ± 3.1	59.9 ± 12.2	62.5 ± 11.7	58.6 ± 1.1	57.0 ± 1.0

did not undergo update for a stretch of 50,000 epochs (pattern presentations). Training was conducted until either all patterns were correctly classified or 500 training epochs were reached for TP and BCP. In the case of TP, a heuristic alteration of the initial temperature was performed after each epoch (as suggested in [Burgess, 1994]).

Table 3.2 and 3.3 report the average accuracy and the standard deviation over 25 runs of the three algorithms with independent and WTA training strategy, respectively. Datasets without a test set have '–'s in the column for testing accuracy. Dynamic reduction of the learning rate was performed in the loss minimization routine for the WTA BCP. If the cumulative loss diverged over 5 consecutive epochs of the loss minimization routine the learning rate was decreased to 0.95 times its current value (with the initial learning rate set to 1.0). Θ is randomly initialized and α are randomly initialized to integer values between 1 and 4.

As we can see from Table 3.2 and 3.3, given enough training time the three algorithms are comparable in general. However, each algorithm outperforms the others on some datasets. For almost all multi-category datasets, WTA strategy gave higher accuracies (except the **wine** dataset for BCP).

Table 3.3 Classification accuracy (WTA).

	RP		TP		BCP	
Dataset	*train*	*test*	*train*	*test*	*train*	*test*
balance	92.1 ± 0.9	89.2 ± 1.4	91.8 ± 1.1	89.5 ± 0.8	92.3 ± 0.3	89.9 ± 0.2
glass	56.5 ± 4.5	48.2 ± 6.9	58.8 ± 3.9	43.2 ± 3.6	49.0 ± 0.5	51.2 ± 1.9
r5	67.9 ± 1.9	–	76.3 ± 1.8	–	66.6 ± 1.5	–
soybean	100 ± 0.0	98.9 ± 2.7	100 ± 0.0	96.9 ± 4.2	100 ± 0.0	100 ± 0.0
wine	91.7 ± 2.4	95.7 ± 1.2	91.5 ± 0.4	93.5 ± 1.1	70.2 ± 0.4	81.3 ± 0.6
WTA-sep	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0

3.4.2 Training Time

In the previous set of experiments virtually unlimited training time is allowed, but it does not guarantee monotonic increase of training accuracy. We have compared the relative speeds of the three algorithms by measuring the total CPU time in seconds taken by each algorithm to reach 50%, 60%, 70%, 80% and 90% classification accuracy on the training set. Tables 3.4-3.9 show the average training times and the standard deviations needed to achieve the above accuracy milestones for 25 runs of each dataset. The total time represents the CPU time taken to achieve either 100% classification accuracy (on separable datasets) or the total time to complete the maximum epochs allowed for training. A '-' in a column indicates that the

Table 3.4 Training time for RP (independent).

Dataset	50%	60%	70%	80%	90%	total time
balance	0.04 ± 0.01	0.05 ± 0.01	0.06 ± 0.02	0.09 ± 0.03	–	4.03 ± 1.00
concentric	0.05 ± 0.00	0.07 ± 0.02	–	–	–	65.63 ± 60.62
glass	4.94 ± 0.00	–	–	–	–	12.77 ± 3.40
ionosphere	0.03 ± 0.01	0.05 ± 0.01	0.07 ± 0.02	0.13 ± 0.05	1.06 ± 0.62	6.70 ± 1.23
liver	0.01 ± 0.01	0.06 ± 0.04	0.82 ± 0.83	–	–	3.78 ± 1.14
p7	0.02 ± 0.02	1.16 ± 1.34	–	–	–	4.12 ± 0.97
pima	0.03 ± 0.01	0.04 ± 0.02	3.77 ± 2.79	–	–	4.92 ± 1.65
r5	0.33 ± 0.50	–	–	–	–	5.83 ± 1.48
sep	0.01 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.01 ± 0.01	0.02 ± 0.01	0.35 ± 0.17
soybean	0.02 ± 0.01	0.03 ± 0.01	0.03 ± 0.01	0.05 ± 0.02	0.06 ± 0.02	0.08 ± 0.02
wdbc	0.06 ± 0.02	0.10 ± 0.01	0.13 ± 0.03	0.11 ± 0.03	0.21 ± 0.07	8.82 ± 4.91
wine	0.23 ± 0.25	0.59 ± 0.87	10.14 ± 1.91	20.62 ± 0.00	–	15.71 ± 5.38
WTA-sep	0.13 ± 0.06	0.40 ± 0.22	1.45 ± 0.51	–	–	4.61 ± 0.85

Table 3.5 Training time for TP (independent).

Dataset	50%	60%	70%	80%	90%	total time
balance	0.26 ± 0.31	0.25 ± 0.23	0.17 ± 0.18	0.26 ± 0.22	–	48.48 ± 33.69
concentric	0.08 ± 0.02	0.08 ± 0.03	–	–	–	905.99 ± 1001.31
glass	5.04 ± 2.35	–	–	–	–	59.29 ± 30.02
ionosphere	0.04 ± 0.02	0.06 ± 0.02	0.11 ± 0.03	0.20 ± 0.06	1.00 ± 0.52	148.13 ± 27.32
liver	0.05 ± 0.06	0.23 ± 0.12	0.81 ± 1.13	–	–	30.50 ± 12.68
p7	0.01 ± 0.00	2.32 ± 1.24	–	–	–	22.70 ± 2.80
pima	0.17 ± 0.30	0.23 ± 0.30	8.71 ± 3.84	–	–	79.87 ± 28.33
r5	0.23 ± 0.21	1.27 ± 0.07	–	–	–	9.15 ± 3.52
sep	0.01 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.02 ± 0.01	0.03 ± 0.01	0.39 ± 0.13
soybean	0.04 ± 0.01	0.05 ± 0.01	0.05 ± 0.01	0.07 ± 0.01	0.08 ± 0.02	0.10 ± 0.02
wdbc	0.30 ± 0.37	2.11 ± 2.08	1.17 ± 0.81	1.57 ± 1.00	1.72 ± 1.18	175.08 ± 36.51
wine	0.29 ± 0.13	0.51 ± 0.49	5.73 ± 0.36	–	–	88.34 ± 4.30
WTA-sep	0.14 ± 0.06	0.38 ± 0.20	0.96 ± 0.35	–	–	7.26 ± 1.68

Table 3.6 Training time for BCP (independent).

Dataset	50%	60%	70%	80%	90%	total time
concentric	0.43 ± 0.01	–	–	–	–	206.97 ± 2.21
ionosphere	–	–	0.04 ± 0.01	0.14 ± 0.08	1.29 ± 0.68	14.08 ± 0.28
liver	–	0.02 ± 0.01	0.28 ± 0.21	–	–	11.54 ± 0.32
p7	0.01 ± 0.00	0.46 ± 0.44	–	–	–	4.16 ± 0.22
pima	–	0.08 ± 0.00	0.42 ± 0.25	–	–	39.04 ± 0.64
sep	–	–	–	–	0.02 ± 0.00	0.18 ± 0.04
wdbc	–	–	–	0.06 ± 0.00	1.65 ± 0.37	25.30 ± 0.52

corresponding level of training was not achieved (i.e., either the training jumped to a higher level of accuracy or the training could achieve only a lower accuracy level). Since the independent BCP training algorithms for datasets with multiple output classes involves independently training each class of patterns, it is not possible to measure the total time to achieve the various accuracy levels.

Theoretically, the variants of the perceptron learning algorithm converge to the optimal classification on linearly non-separable data only when unlimited training time is allowed. However, in practice unlimited training time is not allowable and thus a limited time should be given instead. From the results we observe that in general RP takes the least total time for training. It reaches the highest accuracy much faster than TP and BCP. This can be attributed

Table 3.7 Training time for RP (WTA).

Dataset	50%	60%	70%	80%	90%	total time
balance	0.12 ± 0.03	0.15 ± 0.04	0.20 ± 0.07	0.23 ± 0.07	1.84 ± 1.50	12.54 ± 3.81
glass	3.59 ± 1.92	19.48 ± 4.23	–	–	–	23.94 ± 8.17
r5	0.03 ± 0.02	0.25 ± 0.28	5.09 ± 1.32	–	–	10.80 ± 2.36
soybean	0.03 ± 0.01	0.02 ± 0.01	0.03 ± 0.01	0.04 ± 0.02	0.06 ± 0.02	0.09 ± 0.03
wine	0.14 ± 0.05	0.21 ± 0.12	2.33 ± 0.82	6.14 ± 1.26	13.13 ± 2.71	24.26 ± 3.49
WTA-sep	0.08 ± 0.03	0.12 ± 0.06	0.22 ± 0.07	0.31 ± 0.09	0.57 ± 0.17	1.14 ± 0.28

Table 3.8 Training time for TP (WTA).

Dataset	50%	60%	70%	80%	90%	total time
balance	0.43 ± 0.54	1.92 ± 1.04	1.46 ± 1.08	1.50 ± 1.08	2.96 ± 1.63	112.16 ± 51.20
glass	4.51 ± 3.19	17.09 ± 0.87	–	–	–	137.21 ± 22.79
r5	0.03 ± 0.02	0.22 ± 0.43	1.60 ± 0.68	–	–	75.75 ± 1.63
soybean	0.06 ± 0.02	0.08 ± 0.03	0.08 ± 0.02	0.12 ± 0.01	0.12 ± 0.02	0.15 ± 0.04
wine	0.55 ± 0.30	0.61 ± 0.24	3.90 ± 1.39	7.11 ± 1.07	9.70 ± 1.25	97.32 ± 10.33
WTA-sep	0.12 ± 0.04	0.18 ± 0.06	0.31 ± 0.07	0.49 ± 0.11	0.75 ± 0.18	1.19 ± 0.20

to the simplicity of the algorithm. BCP on the other hand has the merit of achieving high accuracy very rapidly. The high total training time in the case of BCP is indicative of the time spent in training without any substantial improvement in training accuracy. The quick convergence to high accuracy levels in the BCP can be exploited to rapidly train constructive networks.

Table 3.9 Training time for BCP (WTA).

Dataset	50%	60%	70%	80%	90%	total time
balance	–	–	–	7.92 ± 0.37	15.44 ± 3.75	358.20 ± 11.19
glass	–	–	–	–	–	174.25 ± 2.31
r5	0.40 ± 0.01	0.84 ± 0.33	–	–	–	32.88 ± 0.39
soybean	–	–	–	–	0.13 ± 0.13	0.20 ± 0.34
wine	–	2.43 ± 0.01	3.07 ± 0.19	–	–	130.61 ± 0.51
WTA-sep	–	–	0.62 ± 0.00	0.62 ± 0.01	0.64 ± 0.04	4.00 ± 5.46

3.4.3 Learning Curve

There can be various possibilities in the process of perceptron training. For example, an algorithm can reach a *near-optimal* solution very fast but approach to the optimal solution slowly from there. On the other hand, another algorithm can reach the optimal solution with a constant speed. (For example, we can choose the former for problems that need a *reasonable* solution within a time constraint, and choose the latter for problems that need a *good* solution within a finite time limit). Therefore, studying the bias of perceptron algorithms with respect to the learning speed is of interest. We compared the learning speeds of the algorithms by plotting the learning curves of the algorithms on **ionosphere** and the **pima** datasets. Both datasets are real-world and substantially large and fairly good training accuracies are possible with each of the three algorithms on these datasets. Ten runs were performed with the same parameter settings as described earlier. The training and generalization accuracies were measured at the end of each epoch for each of the three algorithms. Note that for the purpose of this experiment in the case of RP and TP, one epoch was measured as a presentation of l randomly chosen training patterns (where l is the total number of training patterns for the dataset) while in the case of the BCP each epoch involved seeing all the training patterns once. Training was performed for 500 epochs.

Figure 3.8 and 3.9 show the learning curves for the three algorithms. In the case of **ionosphere** both BCP and RP climb to a high level of training accuracy very rapidly. TP performs poorly at the start but stabilizes to a good training and test accuracy toward the end of the training epochs. The **pima** dataset clearly shows the demarcation between the three training algorithms. Here the BCP outperforms both TP and RP in both the training and generalization accuracies. TP starts off poorly but eventually stabilizes to an accuracy level comparable to RP.

3.4.4 The Impact of Perceptron Learning in Constructive Neural Networks

As explained in Section 5.1, a perceptron learning algorithm can not classify a linearly non-separable data [Minsky & Papert, 1969]. *Constructive neural network learning algorithms*

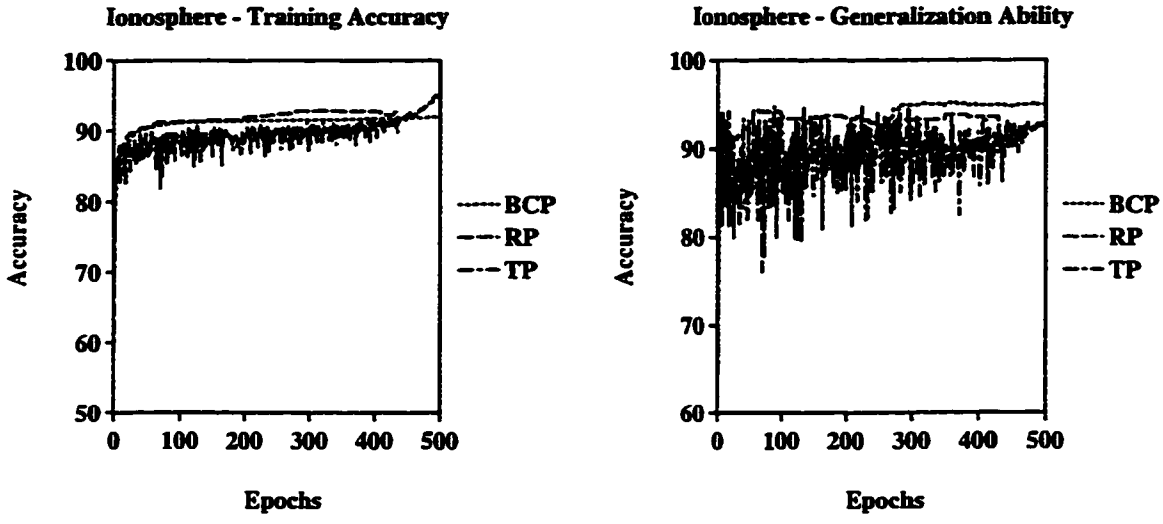


Figure 3.8 Learning curve for ionosphere.

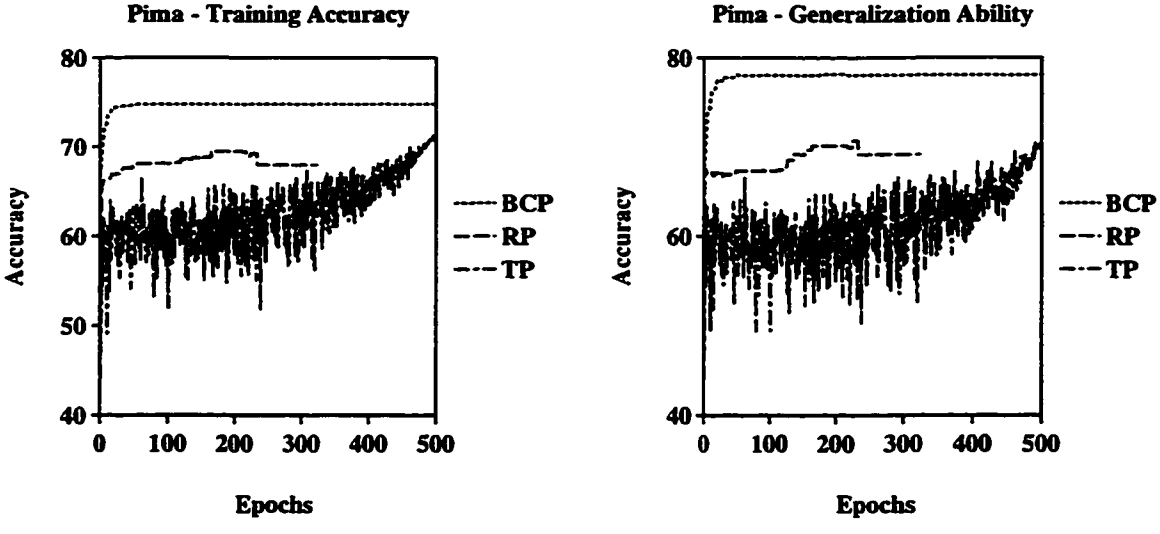


Figure 3.9 Learning curve for pima.

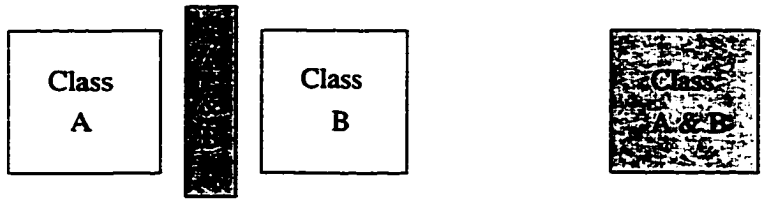
[Gallant, 1993; Honavar & Uhr, 1993; Honavar *et al.*, 1999a] provide a framework for incremental construction of networks. They keep recruiting a set of hidden neurons and setting the connections (i.e., weights) between neurons in a systematic way until the stopping criteria are satisfied (e.g., 100% training accuracy is reached or the number of hidden neurons recruited exceeds some limit). Several constructive learning algorithms appeared in the literature and are shown to guarantee 100% training accuracy theoretically [Mézard & Nadal, 1989; Nadal, 1989; Frean, 1990; Gallant, 1990; Marchand *et al.*, 1990; Burgess, 1994; Yang *et al.*, 1996; Parekh *et al.*, 1997b; Parekh, 1998; Yang *et al.*, 1998b]. Most of them employ a perceptron style weight update procedure (e.g., RP) to determine the weight setting between neurons. The performance of the perceptron algorithm used in the constructive learning algorithm determines the overall performance of classification. In other words, if the perceptron learning algorithm fails to find a proper weight setting for newly recruited neurons, it would not decrease the classification error and thus not converge to 100% training accuracy. Therefore, exploring the bias of a perceptron learning algorithm in the context of constructive learning algorithm is clearly of interest.

The Tiling algorithm [Mézard & Nadal, 1989; Yang *et al.*, 1996; Parekh, 1998] is chosen in our experiments. The Tiling algorithm constructs a strictly layered network of TLUs. The bottom-most layer receives inputs from each of the input neurons. The neurons in each subsequent layer receive inputs from those in the layer immediately below itself. Each layer maintains M *master neurons* for M output classes. The network construction procedure ensures that the master neurons in a given layer correctly classify more patterns than the master neurons of the previous layer. Each layer maintains a set of ancillary neurons that are added and trained to ensure a *faithful representation* of the training patterns. The *faithfulness* criterion states that no two training examples belonging to different classes should produce identical output at any given layer. See chapter 4 for a detailed description on Tiling algorithm. The Tiling algorithm is shown to outperform other constructive learning algorithms in various datasets [Parekh, 1998]. The reason is most likely due to the fact that the Tiling algorithm trains neurons on progressively smaller subset of the entire training set.

Three artificial datasets are used to study the internal bias of RP, TP and BCP:

- **concentric**: concentric circles dataset in Table 3.1.
- **D1**: two well-formed clusters of two classes with a region of mixed patterns belonging to different classes (see Figure 3.10(a)). There are 250 patterns in both training and test sets.
- **D2**: totally randomly generated patterns (see Figure 3.10(b)). There are 250 patterns in both training and test sets.

In Figure 3.10, white boxes denote patterns belonging to a single class and shaded boxes indicate patterns belonging to both classes (A and B).



(a) D1: Marginally non-separable data (b) D2: Random data

Figure 3.10 Two artificial datasets.

Table 3.10 shows the average performance of 10 runs of the perceptron algorithms on the three artificial datasets. Table 3.11 shows the average performance of 10 runs of the Tiling algorithm combined with the perceptron algorithms for the three artificial datasets (Tiling-RP, Tiling-TP, and Tiling-BCP). Here, *success* is the number of runs that succeeded to converge.

Table 3.10 Performance in a single layer perceptron

	RP		TP		BCP	
Dataset	*train*	*test*	*train*	*test*	*train*	*test*
concentric	62.7	64.0	62.7	64.0	55.9	53.1
D1	91.9	90.7	91.0	90.8	92.1	90.2
D2	55.7	53.3	53.3	47.6	56.8	56.8

Table 3.11 Performance in Tiling algorithm

	Tiling-RP			Tiling-TP			Tiling-BCP		
Dataset	*success*	*train*	*test*	*success*	*train*	*test*	*success*	*train*	*test*
concentric	0	-	-	0	-	-	10	100	99.1
D1	3	100	89.2	10	100	90.0	10	100	89.3
D2	9	100	50.5	9	100	50.0	10	100	51.7

Table 3.10 shows comparable accuracy between the three algorithms in **D1** and **D2**. However, BCP gave lower accuracy for **concentric** since the barycenters of patterns in both classes are very close.

The performance of Tiling algorithm with different perceptron algorithms is different from that of perceptron algorithms used alone. In particular, Tiling-BCP always converged to networks with 100% training accuracy while Tiling-RP and Tiling-TP did not converge at all on **concentric**. On **D1** and **D2**, the Tiling-BCP always converged to networks with 100% training accuracy while Tiling-RP and Tiling-TP failed to converge in several runs. All three algorithms gave comparable generalization accuracy. This shows that Tiling-BCP outperforms Tiling-RP and Tiling-TP though there was not much difference in single layer perceptron learning.

3.5 Summary and Discussion

Perceptrons are simple computing elements inspired by a biological neuron. A perceptron is capable of classifying a linearly separable set of patterns. The perceptron learning algorithm finds a proper weight setting to classify such patterns. However, the perceptron learning algorithm can not classify linearly non-separable datasets, and its behavior is unpredictable. A number of variants of the perceptron learning algorithm have been proposed to optimize the performance for linearly non-separable datasets.

Three most commonly used variants of perceptron learning algorithms (pocket algorithm [Gallant, 1993], thermal perceptron [Frean, 1992], and barycentric correction procedure [Poulard, 1995]) are compared experimentally on a collection of benchmark datasets. Both independent and winner-take-all strategies were designed for the algorithms and evaluated on multiple out-

put classes.

The three algorithms were comparable in terms of training accuracy on the datasets used for comparison though some algorithm performed better than others on some datasets. The difference in the accuracy was large for some datasets due to the inductive bias of each algorithm. For example, BCP performed poorly on **concentric** dataset which produces similar barycenters for each class. On the contrary, BCP performed even better than RP and TP on the difficult **pima** dataset. As expected, WTA training gave higher accuracies than independent training.

BCP approached the best accuracy very fast while RP and TP required sufficient time. Because of its simplicity, RP was the fastest given the stopping criteria set up for each algorithm.

BCP's fast convergence was also verified by the learning curves on **ionosphere** and **pima** datasets. Both BCP and TP showed nice performance of continuously increasing generalization accuracies during learning.

Preliminary experiments on three artificial datasets show that BCP outperforms RP and TP in a constructive learning algorithm.

Some avenues for future research include:

- Each variant of the perceptron learning algorithm has its own inductive bias. In other words, some algorithm can outperform others for certain kind of tasks. Exploiting the bias and choosing the right algorithm for a dataset will be of significant importance.

- Multilayer perceptrons are necessary to obtain 100% training accuracy for linearly non-separable data. Constructive neural network learning algorithms generate networks of multiple layers of perceptrons dynamically for this purpose. Most of the constructive learning algorithms rely on a perceptron style weight update procedure to find the weight setting for generated neurons. The performance of a constructive learning algorithm heavily depends on the perceptron learning algorithm it employs. Each perceptron learning algorithm has its own inductive bias. Exploiting the bias and choosing the right algorithm for a dataset will be of significant importance. Further experiments with various datasets combined with several constructive learning algorithms are necessary for

a thorough study of the inductive bias of each perceptron learning algorithm. Chapter 4 introduces several constructive neural network learning algorithms for real-valued patterns with multiple classes.

4 CONSTRUCTIVE LEARNING ALGORITHMS FOR NETWORKS OF PERCEPTRONS

4.1 Introduction

Constructive (or *generative*) neural network algorithms are not restricted to a priori chosen network architectures. Instead, they attempt to dynamically discover an adequate network architecture for a given task. Some of the motivations for using constructive neural network learning algorithms include [Honavar, 1990; Honavar & Uhr, 1993; Parekh, 1998]:

- *Limitations of learning by weight modification alone within an a priori fixed network architecture:*

  Learning by weight modification involves searching the weight space for an acceptable solution that satisfies the desired performance criterion (e.g., classification error). To be successful, such a solution must lie within the weight space being searched, and the search procedure employed must be able to locate it. If the user does not have adequate problem-specific knowledge that could be used to choose an appropriate network architecture, the process can try *ad hoc* choices of network architectures, which leads to trial-and-errors. Constructive algorithms can potentially offer a way around this problem by extending the search for a solution to the space of network topologies in a controlled fashion.

- *Complexity of the network and the classification task:*

  It is desirable that a learning algorithm construct networks whose complexity (in terms of relevant criteria such as number of nodes, number of links, connectivity, etc.) is commensurate with the intrinsic complexity of the classification task (implicitly specified by the training data). Everything else being equal, the more compact the network, the

more likely it is that it exhibits better generalization properties. In addition, smaller networks yield efficient hardware implementations. Constructive algorithms can potentially discover near-minimal networks for correct classification of a given dataset.

- *Estimation of expected case complexity of pattern classification tasks:*

  Many pattern classification tasks are known to be computationally hard. However, little is known about the *expected* case complexity of classification tasks that are encountered, and successfully solved, by living systems - primarily because it is difficult to mathematically characterize the properties of such problem instances. Constructive algorithms, if successful, can provide useful empirical estimates of expected case complexity of real-world pattern classification tasks.

- *Trade-offs among performance measures:*

  Different constructive learning algorithms offer natural means of trading off certain performance measures against others. For instance, some algorithm can yield higher generalization accuracy and/or smaller network size but longer training time. Others can converge to the solution faster but with lower generalization accuracy and/or larger network size.

- *Incorporation of prior knowledge:*

  Constructive algorithms provide a natural framework for incorporating problem-specific knowledge into the initial network configuration. Knowledge refinement can then be performed using new training examples.

Two prominent approaches to constructive learning include *growing algorithms* and *shrinking algorithms*:

- *Growing algorithms*:

  They start with an initial network composed of input and output neurons. A set of hidden neurons are then added into the network iteratively based on some criteria [Mézard & Nadal, 1989; Gallant, 1990; Frean, 1990; Marchand *et al.*, 1990; Burgess, 1994; Yang *et al.*, 1998b]. For instance, when the current network does not yield improved training

accuracy, new hidden neurons are added, and then trained. This process of addition and training of new hidden neurons is repeated until some suitably selected stopping criteria is satisfied (e.g., the training accuracy increased up to some level or the number of hidden neurons exceed some limit).

- *Shrinking algorithms*:

  They are exactly the opposite of growing algorithms. They start with sufficient number of hidden neurons in the initial network. A set of hidden neurons are then deleted from the network based on some criteria. For example, if some hidden neurons are determined to be redundant or irrelevant, they are eliminated [Mozer & Smolensky, 1989; Parekh *et al.*, 1997c]. In addition, the error function can be modified to include a penalty term for weight decay, and the weights of insignificant magnitude are eliminated [Hanson & Pratt, 1991; Le Cun *et al.*, 1990; Weigend *et al.*, 1991].

In both cases, there should be a well-defined procedures of growing or shrinking the networks. For instance, it should be specified that where new neurons are added, how they are interconnected, which set of training patterns are to be used to train newly added neurons, what learning algorithm is to be used, which neurons are deleted on what basis, and the like. In this dissertation, we focus on growing algorithms.

A number of growing constructive learning algorithms have been proposed in the literature. They are classified into two groups based on the type of the neurons recruited: sigmoid or perceptron.

[Ash, 1989; Fahlman & Lebiere, 1990; Hirose *et al.*, 1991] proposed constructive neural network learning algorithms with sigmoid neurons. They used training algorithms for neurons similar to the backpropagation algorithm. A constructive learning algorithm using radial basis function was also proposed in [Roy *et al.*, 1995].

[Mézard & Nadal, 1989; Nadal, 1989; Frean, 1990; Gallant, 1990; Golea & Marchand, 1990; Marchand *et al.*, 1990; Bose & Garga, 1993; Burgess, 1994; Campbell & Vicente, 1995; Deffuant, 1995] proposed constructive neural network learning algorithms with perceptrons. Such networks offer significant representational and computational efficiencies and simpler digital

hardware realizations than their continuous counterparts. In addition, they are guaranteed to converge to zero classification errors on all finite and non-contradictory datasets using a perceptron learning algorithm.

There have been also several approaches in the design of neural networks using genetic algorithms [Kitano, 1990; Andersen & Tsoi, 1993]. They produce the best neural network based on some fitness function using either sigmoid neurons or perceptrons as hidden neurons.

Among the various constructive neural network learning algorithms with perceptrons for 2-category pattern classification tasks proposed in the literature, the *Tower* [Nadal, 1989; Gallant, 1990], *Pyramid* [Gallant, 1990], *Tiling* [Mézard & Nadal, 1989], *Upstart* [Frean, 1990], *Perceptron Cascade* [Burgess, 1994], and *Sequential* [Marchand *et al.*, 1990] algorithms are studied in this dissertation. With the exception of the *Sequential* learning algorithm, constructive learning algorithms are based on the idea of transforming the task of determining the necessary network topology and weights to two subtasks:

- Incremental addition of one or more perceptrons to the network when the existing network topology fails to achieve the desired classification accuracy on the training set.
- Training the added perceptrons using some variant of the perceptron training algorithm.

In the case of the *Sequential* learning algorithm, hidden neurons are added and trained by an appropriate weight training rule to exclude as many patterns belonging to the same class as possible from the currently unexcluded patterns.

The constructive algorithms differ in terms of their choices regarding: restrictions on input representation (e.g., binary, bipolar, or real-valued inputs); when to add a neuron; where to add a neuron; connectivity of the added neuron; weight initialization for the added neuron; how to train the added neuron (or a subnetwork affected by the addition); and so on. The interested reader is referred to [Chen *et al.*, 1995] for an analysis (in geometrical terms) of the decision boundaries generated by some of these constructive learning algorithms. Each of these algorithms can be shown to converge to networks which yield zero classification errors on any given training set wherein the patterns belong to one of two classes (i.e., 2-category classification). The convergence proof of the *Sequential* learning algorithm is based on the

ability of the perceptron weight training algorithm to exclude at least one formerly unexcluded pattern from the training set each time a new hidden neuron is trained. In the case of the other algorithms the convergence proof is based on the ability of the perceptron weight training algorithm to find a weight setting for each newly added neuron or neurons such that the number of pattern misclassifications is reduced by at least one each time a neuron (or a set of neurons) is added and trained and the network's outputs are recomputed. We will refer to such a perceptron weight training algorithm as $\mathcal{A}$ and assume that it will correspond to an appropriate choice depending on the constructive algorithm being considered. In practice, the performance of the constructive algorithm depends partly on the choice of $\mathcal{A}$ and its ability to find weight settings that will reduce the total number of misclassifications (or to exclude at least one formerly unexcluded pattern from the training set) each time new neurons are added to the network and trained. Some possible choices for $\mathcal{A}$ when the desired task is to maximize classification accuracy are the *pocket algorithm* [Gallant, 1990; Gallant, 1993], the *thermal perceptron algorithm* [Frean, 1992], and the *barycentric correction procedure* [Poulard, 1995] as introduced in chapter 3. A variant of the *barycentric correction procedure* can be used to efficiently exclude patterns as desired by the *Sequential* learning algorithm.

Pattern classification tasks often require assigning patterns to one of M ($M > 2$) classes. Although in principle, an M-category classification task can be reduced to an equivalent set of M 2-category classification tasks (each with its own training set constructed from the given M-category training set), a better approach might be one that takes into account the inter-relationships between the M output classes. For instance, the knowledge of membership of a pattern $\mathbf{X}^p$ in category Ψ_i can be used by the learning algorithm to effectively rule out its membership in a different category Ψ_j ($j \neq i$) and any internal representations learned in inducing the structure of Ψ_i can therefore be exploited in inducing the structure of some other category Ψ_j ($j \neq i$). In the case of most constructive learning algorithms, extensions to multiple output classes have not been explored. In other cases, only some preliminary ideas (not supported by detailed theoretical or experimental analysis) for possible multi-category extensions of 2-category algorithms are available in the literature.

Additionally, practical classification tasks often involve patterns with real-valued attributes. The perceptron weight training algorithms like the *pocket algorithm*, *thermal perceptron algorithm*, and *barycentric correction procedure* do handle patterns with real-valued attributes. However, extensions of the constructive learning algorithms to handle patterns with real-valued attributes have only been studied for the *Upstart* [Saffery & Thornton, 1991] and the *Perceptron Cascade* [Burgess, 1994] algorithms.

Against this background, this chapter revisits the extended versions of the *Tower* (Section 4.2), *Pyramid* (Section 4.3), *Tiling* (Section 4.6), *Upstart* (Section 4.4). *Perceptron Cascade* (Section 4.5). and *Sequential* (Section 4.7) algorithms for multi-category pattern classification with real-valued patterns studied in [Parekh, 1998]. Detailed algorithmic description, convergence proof. and performance evaluations of these algorithms are given in [Parekh, 1998]. Section 4.8 concludes with summary and future research directions.

4.2 Tower Algorithm

The 2-category *Tower* algorithm [Nadal, 1989: Gallant, 1990] constructs a tower of perceptrons. The bottom-most neuron receives inputs from each of the N input neurons. The tower is built by successively adding neurons to the network and training them using $\mathcal{A}$ until the desired classification accuracy is achieved. Each newly added neuron receives input from each of the N input neurons and the output of the neuron immediately below itself and takes over the role of the network's output.

To handle patterns with real valued attributes it is necessary to consider the projection of the patterns onto a parabolic surface [Burgess, 1994]. The extension of the 2-category *Tower* algorithm to deal with multiple (M) output categories is accomplished by simply adding M neurons each time a new layer is added to the tower. Each neuron in the newly added layer (which then serves as the network's output layer) receives inputs from the $N+1$ input neurons (including the extra input of projection) as well as the M neurons in the preceding layer (if one exists). The newly added neurons are trained using $\mathcal{A}$. A *Tower* network with two hidden layers is shown in Figure 4.1.

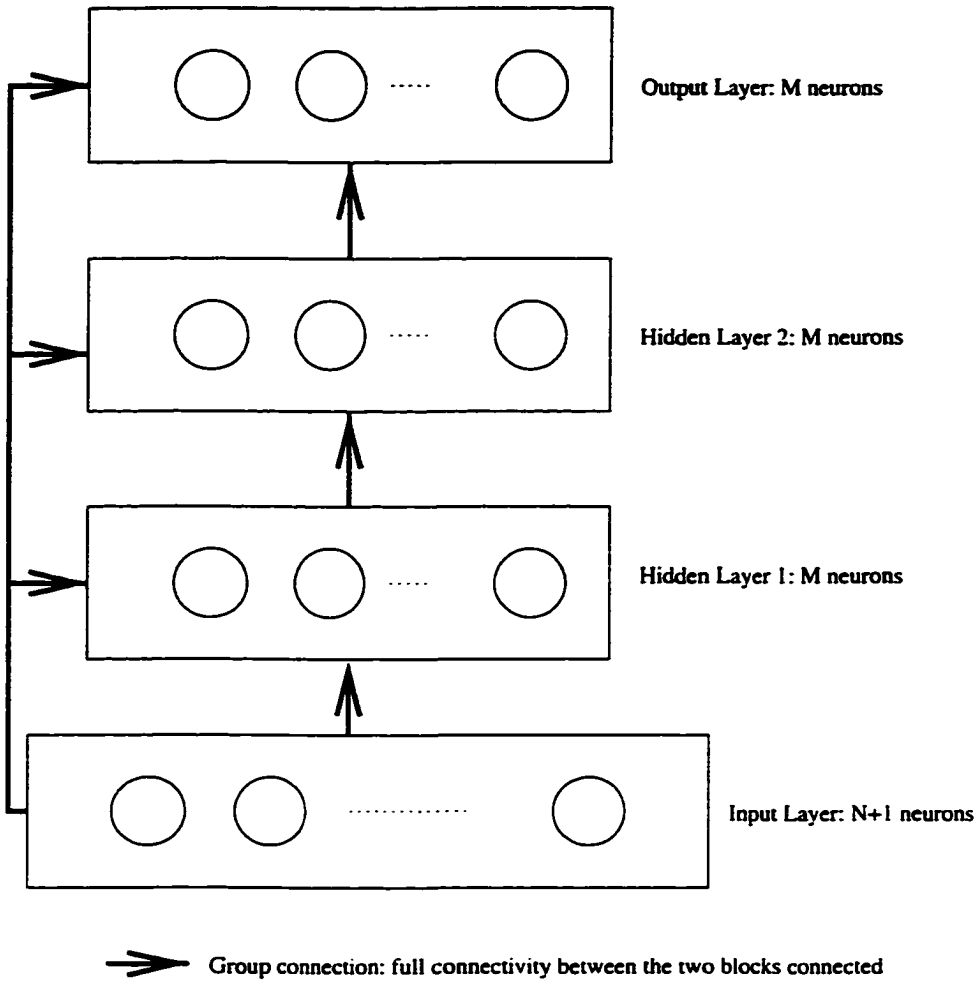


Figure 4.1 Multi-category Tower network.

4.3 Pyramid Algorithm

The 2-category *Pyramid* algorithm [Gallant. 1990] constructs a network in a manner similar to the *Tower* algorithm, except that each newly added neuron receives input from each of the N input neurons as well as the outputs of all the neurons in each of the preceding layers. The newly added neuron becomes the output of the network. As in the case of the *Tower* algorithm, the extension of the 2-category *Pyramid* algorithm to handle M output categories and real-valued pattern attributes is quite straightforward. Each pattern is modified by appending the extra attribute (X^p_{N+1}). Each newly added layer of M neurons receives inputs from the $N+1$

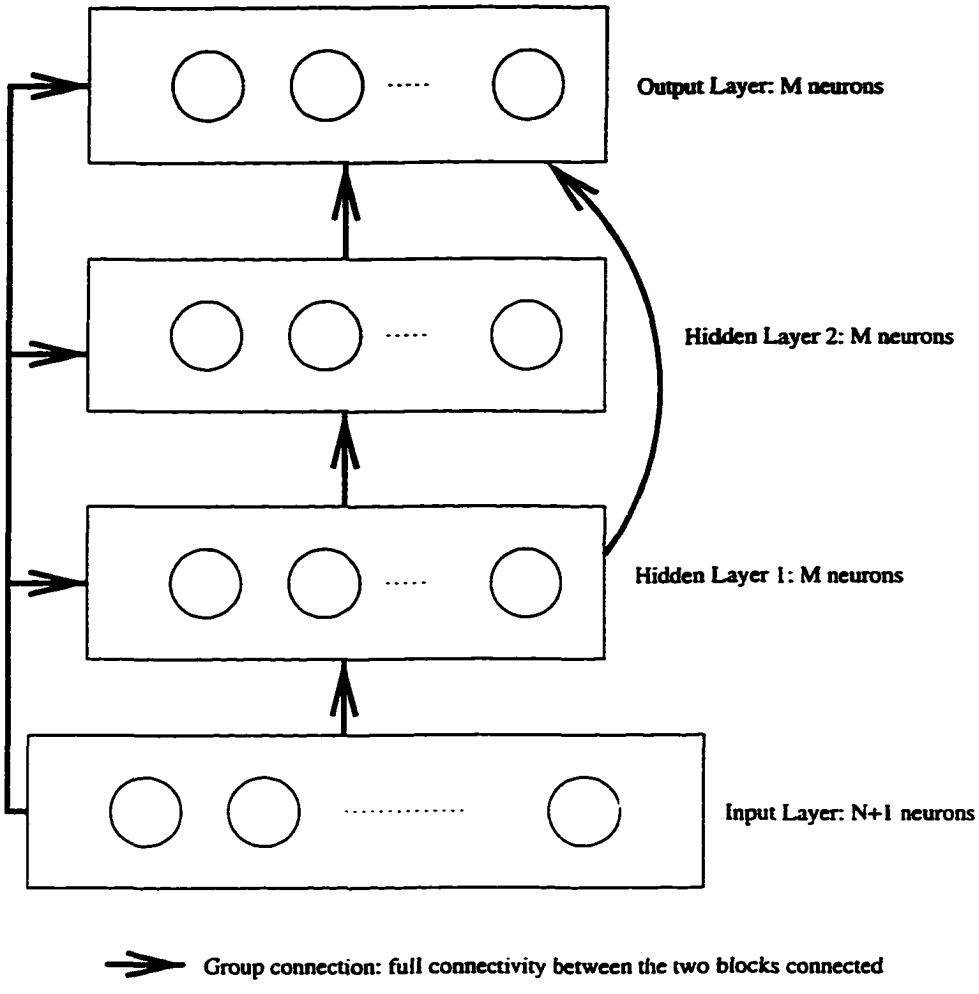


Figure 4.2 Multi-category Pyramid network.

input neurons and the outputs of each neuron in each of the previously added layers. A *Pyramid* network with two hidden layers is shown in Figure 4.2.

4.4 Upstart Algorithm

The 2-category *Upstart* algorithm [Frean, 1990] constructs a binary tree of threshold neurons. A simple extension of this idea to deal with M output categories would be to construct M independent binary trees (one for each output class). This approach fails to exploit the inter-relationships that might exist between the different outputs. We therefore follow an al-

ternative approach [Frean, 1990] using a single hidden layer instead of a binary tree. Again, to handle patterns with real-valued attributes we consider the projection of the pattern vectors[1].

The extension of the *Upstart* algorithm to handle multiple output categories is described as follows. First, an output layer of M neurons is trained using the algorithm $\mathcal{A}$. If all the patterns are correctly classified, the procedure terminates without the addition of any hidden neurons. If that is not the case, the output neuron (L_k) that makes the most number of errors (in the sense $C_k^p \neq O_{L_k}^p$) is identified. Depending on whether the neuron k is *wrongly-on* (i.e., $C_k^p = 0, O_{L_k}^p = 1$) or *wrongly-off* (i.e., $C_k^p = 1, O_{L_k}^p = 0$) more often, a wrongly-on corrector daughter (X) or a wrongly-off corrector daughter (Y) is added to the hidden layer and trained to correct some errors of neuron L_k. For each pattern $\mathbf{X}^p$ in the training set, the target outputs (C_X^p and C_Y^p) for the X and Y daughters are determined as follows:

- If $C_k^p = 0$ and $O_{L_k}^p = 0$ then $C_X^p = 0$, $C_Y^p = 0$.
- If $C_k^p = 0$ and $O_{L_k}^p = 1$ then $C_X^p = 1$, $C_Y^p = 0$.
- If $C_k^p = 1$ and $O_{L_k}^p = 0$ then $C_X^p = 0$, $C_Y^p = 1$.
- If $C_k^p = 1$ and $O_{L_k}^p = 1$ then $C_X^p = 0$, $C_Y^p = 0$.

The daughter is trained using the algorithm $\mathcal{A}$. The daughter is connected to each neuron in the output layer and its weights are frozen. Then the output weights are retrained. The resulting network is shown in Figure 4.3.

4.5 Perceptron Cascade Algorithm

The *Perceptron Cascade* algorithm [Burgess, 1994] draws on the ideas used in the *Upstart* algorithm and constructs a neural network that is topologically similar to the one built by the *Cascade correlation algorithm* [Fahlman & Lebiere, 1990]. However, unlike the *Cascade correlation algorithm* the *Perceptron Cascade* algorithm uses perceptrons. Initially an output neuron is trained using the algorithm $\mathcal{A}$. If the output neuron does not correctly classify the

[1] An extension of the *Upstart* algorithm to handle patterns with real valued attributes using stereographic projection was originally proposed by [Saffery & Thornton, 1991].

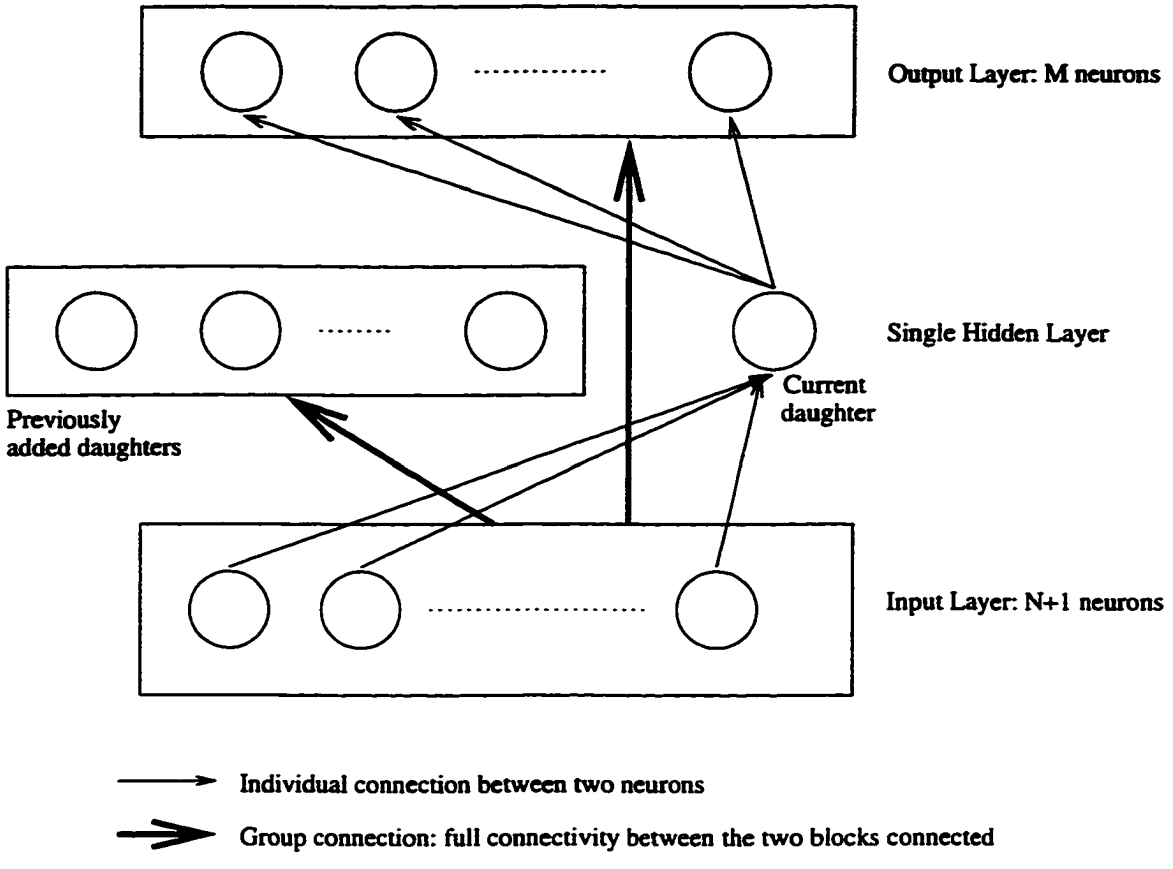


Figure 4.3 Multi-category Upstart network.

training set, a daughter neuron (wrongly-on or wrongly-off as desired) is added and trained to correct some of the errors. The daughter neuron receives inputs from each of the input neurons and from each of the previously added daughters. As shown in Figure 4.4 each daughter neuron is added to a new hidden layer during the construction of the *Perceptron Cascade* network. The targets for the daughter are determined exactly as in the case of the *Upstart* network.

The extension of the *Perceptron Cascade* algorithm to M output classes is relatively straight forward. First, an output layer of M neurons is trained. If all the patterns are correctly classified, the procedure terminates without the addition of any hidden neurons. If that is not the case, the output neuron, L_k, that makes the largest number of errors (in the sense that $C_k^p \neq O_{L_k}^p$) is identified and a daughter neuron (an X daughter if the neuron is wrongly-on more often or a Y daughter if the neuron is wrongly-off more often) is added to a new hidden

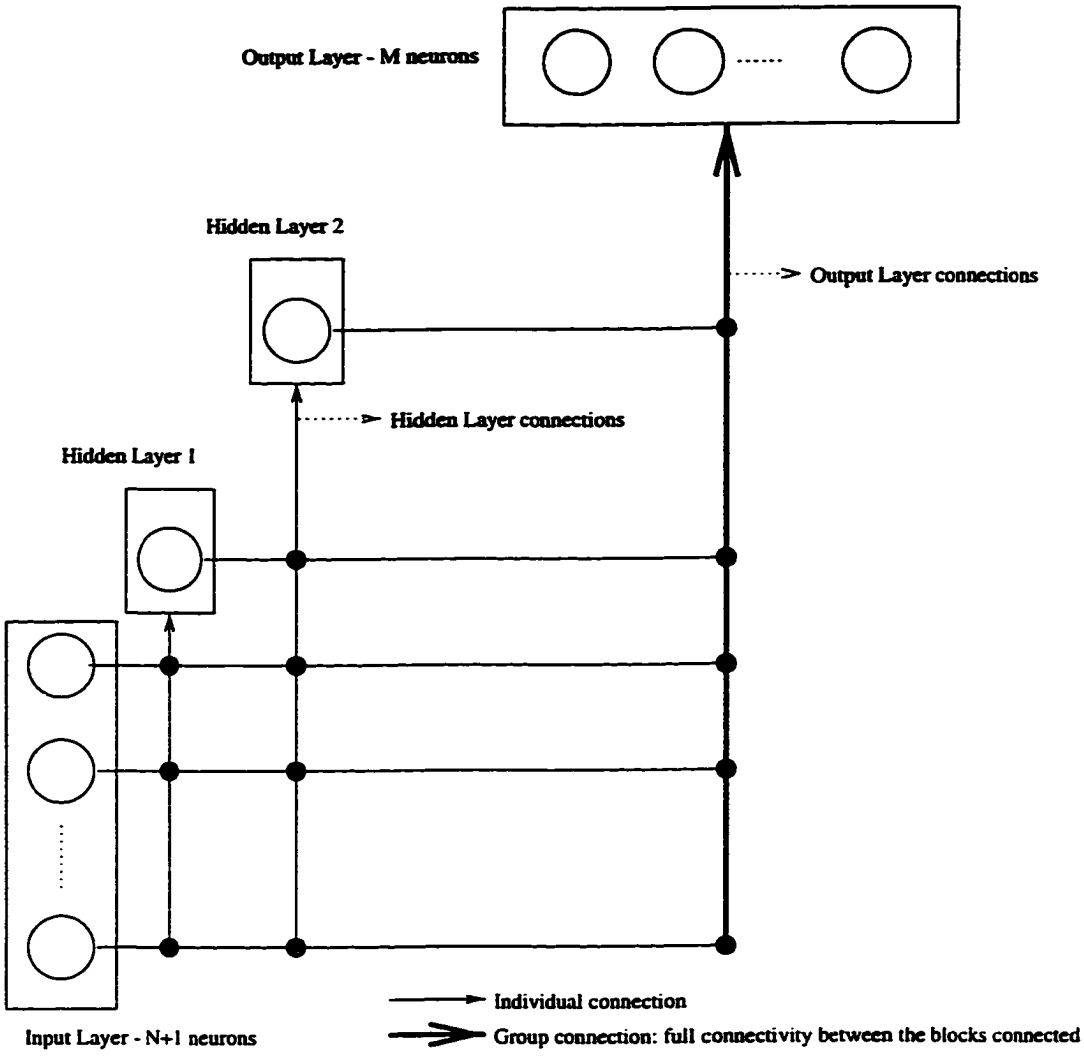


Figure 4.4 Multi-category Perceptron Cascade network.

layer and trained to correct some of the errors made by the output neurons. For each pattern $\mathbf{X}^p$ in the training set, the target outputs for the daughter neuron are determined as in the *Upstart* algorithm. The daughter receives its inputs from each of the input neurons and from the outputs of each of the previously added daughters. After the daughter is trained it is connected to each of the M output neurons and the output weights are retrained. Figure 4.4 shows the construction of a *Perceptron Cascade* network. The extension to handle real-valued pattern attributes involves taking the projection of the patterns.

4.6 Tiling Algorithm

The *Tiling* algorithm [Mézard & Nadal, 1989] constructs a strictly layered network of threshold neurons. The bottom-most layer receives inputs from each of the N input neurons. The neurons in each subsequent layer receive inputs from those in the layer immediately below itself. Each layer maintains a *master neuron.* The network construction procedure ensures that the master neuron in a given layer correctly classifies more patterns than the master neuron of the previous layer. Ancillary neurons may be added to layers and trained to ensure a *faithful representation* of the training set. The *faithfulness* criterion simply ensures that no two training examples belonging to different classes produce identical output at any given layer. Faithfulness is clearly a necessary condition for convergence in strictly layered networks [Mézard & Nadal, 1989].

The proposed extension to multiple output classes involves constructing layers with M master neurons (one for each of the output classes). Unlike the other algorithms seen before, it is not necessary to take the projection of the input patterns to guarantee convergence for patterns with real-valued attributes. Sets of one or more ancillary neurons are trained at a time in an attempt to make the current layer faithful: Among all the unfaithful output vectors at the current output layer, identify the one that the largest number of input patterns map to. (An output vector is said to be unfaithful if it is generated by input patterns belonging to different classes); Determine the set of patterns that generate the output vector identified; Add a set of k ($1 \leq k \leq M$) ancillary neurons where k is the number of target classes represented in the set of patterns identified, and train them; Repeat these (adding and training ancillary neurons) till the output layer representation of the patterns is faithful. Then, train a new layer M master neurons that are connected to each neuron in the previous layer. If desired classification accuracy is obtained, stop. Otherwise, repeat the process described above. Figure 4.5 shows the construction of a *Tiling* network.

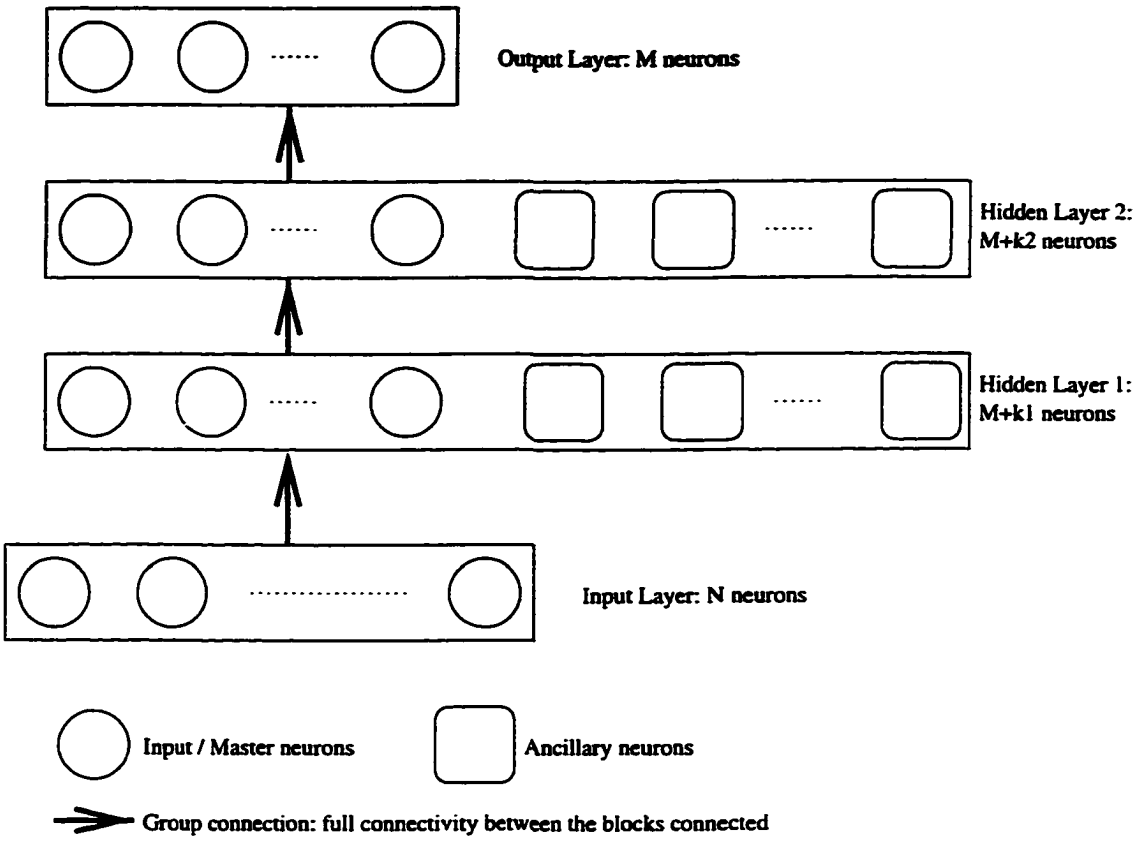


Figure 4.5 Multi-category Tiling network.

4.7 Sequential Learning Algorithm

The *Sequential* learning algorithm [Marchand *et al.*, 1990] offers an alternative scheme for network construction where instead of training neurons to correctly classify a maximal subset of the training patterns, the idea is to train neurons to sequentially exclude patterns belonging to one class from the remaining patterns. The algorithm constructs a two layer network of threshold neurons where the hidden layer neurons are trained to sequentially exclude patterns belonging to a one class. When all the patterns in the training set have been thus excluded, the internal representation of the patterns at the hidden layer is guaranteed to be linearly separable. The weights of the single output layer neuron are fixed to correctly classify all patterns. Recently, Poulard has shown that a variation of the *barycentric correction procedure*

can be used effectively in *Sequential* learning to exclude as many patterns belonging to one class as possible [Poulard, 1995].

The extension of the *Sequential* learning algorithm to multiple output categories follows the same principles as the original version. Using a simple modification of the *barycentric correction procedure*, hidden neurons can be trained to exclude patterns belonging to one of the M classes from the remaining patterns. Once all the patterns in the training set have been excluded by the hidden layer neurons, the output layer with M neurons can be constructed to correctly classify all patterns without going through perceptron weight update procedures [Marchand *et al.*, 1990; Parekh, 1998; Yang *et al.*, 1998b]. As in the case of the *Tiling* algorithm, it is not necessary to consider the projection of the training patterns to prove the convergence for patterns with real-valued attributes. Figure 4.6 depicts a network constructed by *Sequential* learning.

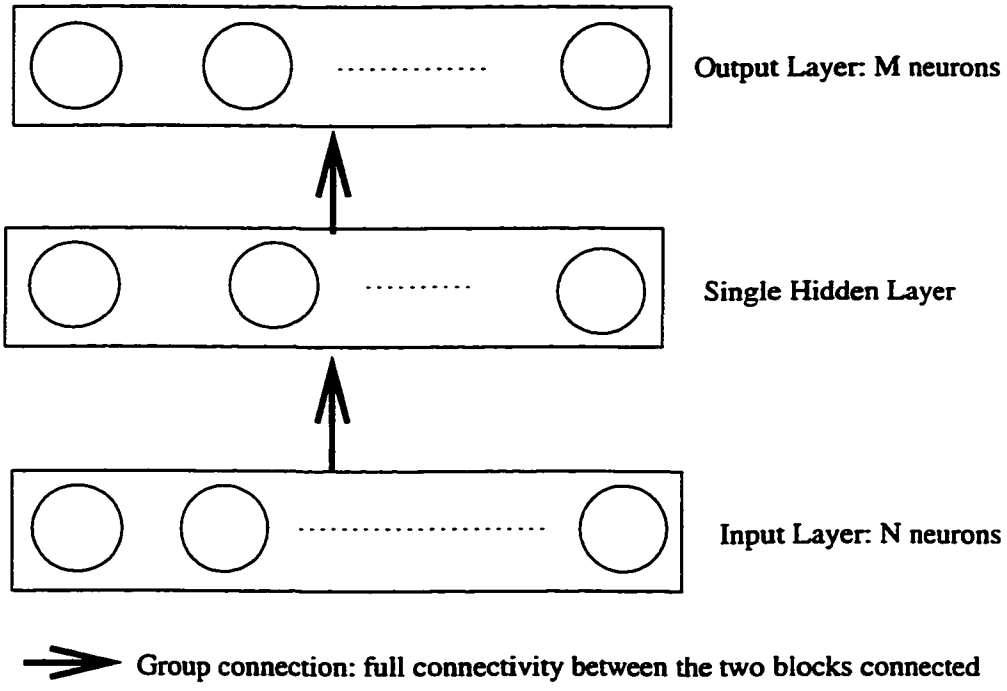


Figure 4.6 Multi-category Sequential network.

4.8 Summary and Discussion

Constructive learning algorithms offer an attractive approach to the automated design of neural networks for pattern classification. In particular, they obviate the need for an *ad hoc, a priori* choice of the network topology. Instead, a network architecture is dynamically determined in proportion to the complexity of the task. Therefore, they have a potential to generate parsimonious, *near-optimal* networks. In addition, they offer natural ways to incorporate prior knowledge into the network to guide learning.

We have focused on a family of such algorithms that incrementally construct networks of perceptrons. Using perceptrons (instead of its continuous counterparts) enables us to use a simple, fast perceptron style weight update procedures without too much parameter finetuning. Although a number of such algorithms have been proposed in the literature, most of them are limited to 2-category pattern classification tasks with binary/bipolar valued input attributes. We extended several existing constructive learning algorithms to handle multi-category classification for patterns having real-valued attributes. All of the algorithms considered were shown to converge to 100% training accuracy on finite and non-contradictory datasets. The convergence of the algorithms relies on the weight training rules used inside. Under the assumption that the weight update procedure finds a proper weight setting successfully, all the algorithms are guaranteed to converge to zero classification error.

Simulation results [Parekh, 1998] demonstrated the usefulness of the constructive neural network learning algorithms in classification tasks. Detailed description of the algorithms, convergence proofs, and experimental results are shown in [Parekh, 1998].

Some avenues for future research include:

- All algorithms shown in this chapter depends on a perceptron learning algorithm and each perceptron algorithm also has its own set of inductive biases implicit in the weight update procedure. A systematic and theoretic characterization of this bias would help us determine proper algorithms (combined with the constructive learning algorithms) for different tasks to maximize the performance.

- **Hybrid network training schemes that dynamically select an appropriate network construction strategy, an appropriate perceptron weight training algorithm, an appropriate output computation strategy and such to obtain locally optimal performance at each step of the classification task are also of interest.**

- Various pre-processing techniques are responsible for transforming the training data in a manner that might simplify the learning task. Normalization and quantization of the training patterns are proposed to handle real-valued patterns [Yang & Honavar. 1996; Parekh, 1998].

- Post-processing techniques such as *pruning* of networks eliminate nodes and connections that do not adversely affect the network's performance. Pruning can potentially overcome the over-fitting problem by yielding more compact networks with superior generalization. In recent work it was demonstrated that the application of simple pruning strategies on the *Tiling* networks leads to substantial reduction in the network sizes [Parekh *et al.*, 1997c].

- The algorithms studied in this chapter generate networks of perceptrons. The behavior and performance of the algorithms heavily depend on the perceptron learning algorithm employed inside. Design of new constructive neural network learning algorithms (possibly with different types of neurons, different learning algorithms, etc.) is of interest. A new constructive learning algorithm, DistAI, is designed and introduced in chapter 5. DistAI is based on inter-pattern distance and does not rely on iterative, time-consuming perceptron learning algorithms.

5 DistAI: AN INTER-PATTERN DISTANCE-BASED CONSTRUCTIVE LEARNING ALGORITHM

5.1 Introduction

Several constructive neural network learning algorithms with perceptrons were introduced in chapter 4. They rely on a perceptron style weight update procedure to find proper weight settings. The time-consuming, iterative nature of the perceptron training algorithm (though considerably faster than the corresponding error guided backpropagation training) often makes the use of such algorithms impractical for very large datasets (e.g., in largescale data mining and knowledge acquisition tasks), especially in applications where reasonably accurate classifiers have to be learned in almost real time. Similarly, hybrid learning systems that use neural network learning as the inner loop of a more complex optimization process (e.g., feature subset selection using a genetic algorithm where evaluation of fitness of a solution requires training a neural network based on a subset of input features represented by the solution and evaluating its classification accuracy [Yang & Honavar, 1997; Yang & Honavar, 1998a; Yang & Honavar, 1998b]) call for a fast neural network training algorithm.

Instance-based learning (IBL) [Aha, 1991; Aha *et al.*, 1991; Turney, 1994; Domingos, 1995] is an approach to learning in which the learning algorithm typically stores some or all of the training examples as prototypes. Each prototype is stored as an ordered pair $(\mathbf{X}, c)$ where $\mathbf{X}$ is a *pattern* represented in some chosen instance language (typically, in the form of a vector of attribute values), and c is the *class* to which $\mathbf{X}$ belongs. Such a system, when used to classify a new pattern $\mathbf{Y}$, uses some *distance function* (e.g., Euclidean distance in the case of real-valued patterns) that computes the distance of $\mathbf{Y}$ from each stored prototype and predicts the classification of $\mathbf{Y}$ using the known classification of the nearest prototype

(or prototypes). Such algorithms, also referred to as *nearest neighbor* techniques have been investigated by researchers in pattern recognition [Cover & Hart, 1967; Diday, 1974; Dasarathy, 1991], case-based reasoning [Stanfill & Waltz, 1986; Cost & Salzberg, 1993; Kolodner, 1993], artificial neural networks [Carpenter & Grossberg, 1991], cognitive psychology [Tversky, 1977; Nosofsky, 1986], and text classification [Salton & McGill, 1983]. Such distance-based techniques are also related to *radial basis function* networks [Broomhead & Lowe, 1988; Powell, 1987; Honavar & Uhr, 1993; Girosi *et al.*, 1995].

Rule induction algorithms [Michalski *et al.*, 1986; Clark & Niblett, 1989] learn sets of rules corresponding to given sets of training examples. They induce a rule to *cover* a subset of training examples. New rules are induced iteratively until all training examples are covered.

We present a new constructive neural network learning algorithm (DistAl), which can be viewed as a variant of the instance-based, nearest-neighbor, radial-basis function-based, and rule induction approaches to pattern classification. DistAl replaces the iterative weight update of neurons that is typically used in constructive learning algorithms by a comparison of pairwise distances among the training patterns. Since the inter-pattern distances are computed only once during the execution of the algorithm our approach achieves a significant speed advantage over other constructive learning algorithms.

The rest of this chapter is organized as follows: Section 5.2 describes DistAl. Section 5.3 presents the results of various experiments designed to evaluate the performance of neural networks trained using DistAl on some benchmark classification problems as well as a document classification task. Section 5.4 concludes with a summary and discussion of some directions for future research.

5.2 DistAl: A New Constructive Learning Algorithm

DistAl differs from other constructive learning algorithms mentioned above in two respects:

- It uses *spherical* threshold units – a variant of the TLU – as hidden neurons. The regions that are defined (or separated) by TLUs are unbounded. This motivates us to use spherical threshold units that cover locally bounded regions [Langley, 1995]. A spherical

threshold neuron i has associated with it a weight vector $\mathbf{W}_i$, two thresholds – $\theta_{i,low}$ and $\theta_{i,high}$, and a suitably defined distance metric d. It computes the distance $d(\mathbf{W}_i, \mathbf{X}^p)$ between a given input pattern $\mathbf{X}^p$ and $\mathbf{W}_i$. The corresponding output $O_i^p = 1$ if $\theta_{i,low} \leq d(\mathbf{W}_i, \mathbf{X}^p) \leq \theta_{i,high}$ and 0 otherwise. The spherical neuron thus identifies a cluster of patterns that lie in the region between two concentric hyperspherical regions. $\mathbf{W}_i$ represents the common center and $\theta_{i,low}$ and $\theta_{i,high}$ respectively represent the boundaries of the two regions.

- DistAl does not use an iterative algorithm for finding the weights and the thresholds. Instead, it computes the inter-pattern distance once between each pair of patterns in the training set and determines the weight values for hidden neurons by a greedy strategy (that attempts to correctly classify as many patterns as possible with the introduction of each new hidden neuron). The weights and thresholds are then set without the computationally expensive iterative process (see section 5.2.2 for details).

The use of one-time inter-pattern distance calculation instead of (usually) iterative, expensive and time-consuming perceptron training procedure makes the proposed algorithm significantly faster than most other constructive learning algorithms. In fact, the time and space complexities of DistAl can be shown to be polynomial in the size of the training set (see section 5.2.6 for details).

5.2.1 Distance Metrics

Each hidden neuron introduced by DistAl essentially represents clusters of patterns that fall in the region bounded by two concentric hyperspherical regions in the pattern space. The weight vector of the neuron defines the center of the hyperspherical regions and the thresholds determine the boundaries of the regions (relative to the choice of the distance metric used). The choice of an appropriate distance metric for the hidden layer neurons is critical to achieving a good performance. Different distance metrics represent different notions of *distance* in the pattern space. They also impose different *inductive biases* [Langley, 1995; Mitchell, 1997] on the learning algorithm. Consequently, many researchers have investigated the use of alternative

distance functions for instance-based learning [Duda & Hart, 1973; Diday, 1974; Salton & McGill, 1983; Batchelor, 1978; Wilson & Martinez, 1997]. The number and distribution of the clusters that result from specific choices of distance functions is a function of the distribution of the patterns as well as the clustering strategy used. Since it is difficult to identify the best distance metric in the absence of knowledge about the distribution of patterns in the pattern space, we chose to explore a number of different distance metrics proposed in the literature.

The distance between two patterns is often skewed by attributes that have high values. *Normalization* of individual attributes overcomes this problem in the distance computation. Normalization can be achieved by dividing each pattern attribute by the *range* of possible values for that attribute, or by 4 times the standard deviation for that attribute [Wilson & Martinez, 1997].

Normalization also allows attributes with nominal and/or missing values to be considered in distance computation. The distance for attributes with nominal values (say with attribute values x and y) is computed as follows [Wilson & Martinez, 1997]:

- *Overlap*: $d_{ol}(x, y) = 0$ if $x = y$; 1 otherwise.
- *Value difference*:

$$d_{vd}(x, y) = \sum_{c=1}^{C} \left| \frac{N_{a,x,c}}{N_{a,x}} - \frac{N_{a,y,c}}{N_{a,y}} \right|^q$$

  where

  - $N_{a,x}(N_{a,y})$: number of patterns in the training set that have value $x(y)$ for attribute a
  - $N_{a,x,c}(N_{a,y,c})$: number of patterns in the training set that have value $x(y)$ for attribute a and output class c
  - C: number of output classes
  - q: a constant (Euclidean: 2, Manhattan: 1)

If there is a missing value in either of the patterns, the distance for that component (of the entire pattern vector) is taken to be 1.

Let $\mathbf{X}^p = [X_1^p, \cdots, X_n^p]$ and $\mathbf{X}^q = [X_1^p, \cdots, X_n^q]$ be two pattern vectors. Let max_i, min_i and σ_i be the maximum, minimum, and the standard deviation of values of the ith attribute of patterns in a dataset, respectively. Then the distance between $\mathbf{X}^p$ and $\mathbf{X}^q$, for different choices of the distance metric d is defined as follows:

1. Range, value-difference based Euclidean (point-to-point):

$$\sqrt{\sum_{i=1}^{n}[(\frac{X_i^p - X_i^q}{max_i - min_i})^2 \ or \ d_{vd}(X_i^p, X_i^q)^2]}$$

2. Range, value-difference based Manhattan (city-block):

$$\sum_{i=1}^{n}[\frac{|X_i^p - X_i^q|}{max_i - min_i} \ or \ d_{vd}(X_i^p, X_i^q)]$$

3. Range, value-difference based Maximum Value:

$$\max_i[\frac{|X_i^p - X_i^q|}{max_i - min_i} \ or \ d_{vd}(X_i^p, X_i^q)]$$

   Similarly, $4 * \sigma_i$ can be used instead of $max_i - min_i$ for standard deviation based metrics, and $d_{ol}(X_i^p, X_i^q)$ can be used instead of $d_{vd}(X_i^p, X_i^q)$ for overlap based metrics in above formulas.

4. Dice coefficient:

$$1 - \frac{2\sum_{i=1}^{n} X_i^p X_i^q}{\sum_{i=1}^{n}(X_i^p)^2 + \sum_{i=1}^{n}(X_i^q)^2}$$

5. Cosine coefficient:

$$1 - \frac{\sum_{i=1}^{n} X_i^p X_i^q}{\sqrt{\sum_{i=1}^{n}(X_i^p)^2 \cdot \sum_{i=1}^{n}(X_i^q)^2}}$$

6. Jaccard coefficient:

$$1 - \frac{\sum_{i=1}^{n} X_i^p X_i^q}{\sum_{i=1}^{n}(X_i^p)^2 + \sum_{i=1}^{n}(X_i^q)^2 - \sum_{i=1}^{n} X_i^p X_i^q}$$

7. Camberra:

$$\sum_{i=1}^{n} \frac{|X_i^p - X_i^q|}{|X_i^p + X_i^q|}$$

Attribute based clustering:

Occasionally, the values of a single attribute between two bounds (say a_{lo} and a_{hi}) might exclusively identify patterns belonging to a particular output class. Thus, a hidden neuron that remembers the name of the attribute a and the two thresholds (a_{lo} and a_{hi}) can be used to form a cluster of patterns belonging to the same class. We use the attribute based comparison in conjunction with the inter-pattern distance based clustering to obtain homogeneous clusters.

5.2.2 Network Construction

DistAI determines a "region" (defined by a spherical hidden neuron) iteratively by a greedy strategy (in terms of the number of training patterns). In other words, it finds a maximal subset of training patterns that can be clustered in a region. The training patterns included in a region are eliminated from further consideration. This set of *ordered* regions are generated until all patterns are included in a region. In testing, a test pattern is checked to see if it lies in a region. The first match is chosen for the classification. If there is no match, the closest region (by a distance metric) is chosen for the classification. Figure 5.1 shows how regions are generated for a dataset of 15 patterns with two classes, O and X. R1, R2, R3, R4 and R5 are determined sequentially to cover 5, 4, 3, 2 and 1 training patterns, respectively. (Another example will be given in Section 5.2.4 with a detailed explanation of network construction).

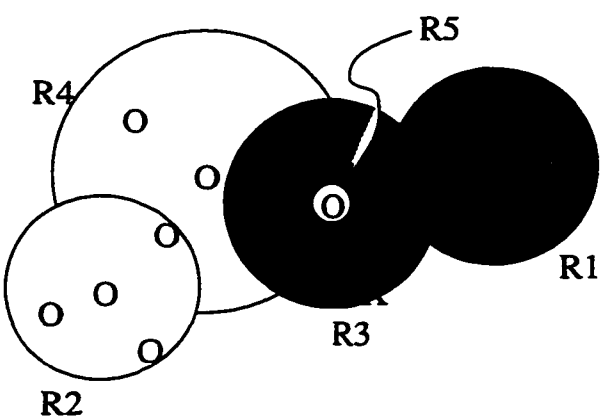


Figure 5.1 Regions induced by DistAI based on the pattern space.

Let $S = \{\mathbf{X}^1, \mathbf{X}^2, \ldots, \mathbf{X}^N\}$ represents the N training patterns. DistAl calculates the pairwise inter-pattern distances for the training set (using the chosen distance metric d) and stores them in the distance matrix $\mathcal{D}$. Each row of $\mathcal{D}$ is sorted in ascending order. Thus, row k of $\mathcal{D}$ corresponds to the training pattern $\mathbf{X}^k$ and the elements $\mathcal{D}[k, i]$ correspond to the distance of $\mathbf{X}^k$ to the other training patterns. $\mathcal{D}[k, 0]$ is the distance to the closest pattern and $\mathcal{D}[k, N]$ is the distance to the farthest pattern from $\mathbf{X}^k$. Simultaneously, the attribute values of the training patterns are stored in $\mathcal{D}'$. $\mathcal{D}'$ is essentially the entire training set with $\mathcal{D}'[k, i]$ representing the ith attribute value of the kth training pattern. Each column (attribute) of $\mathcal{D}'$ is sorted in ascending order.

The key idea behind DistAl is to generate a single layer of hidden neurons each of which separates a subset of patterns in a training set using $\mathcal{D}$ (or $\mathcal{D}'$). Then, they are fully connected to M output TLUs (1 for each output class) in an output layer. The representation of the patterns at the hidden layer is linearly separable [Marchand *et al.*, 1990]. Thus, an iterative perceptron learning rule can be used to train the output weights. However, the output weights can be directly set as follows: The weights between output and hidden neurons are chosen such that each hidden neuron overwhelms the effect of the hidden neurons generated later. If there are a total of h hidden neurons (numbered $1, 2, \ldots, h$ from left to right) then the weight between the output neuron j and the hidden neuron i is set to 2^{h-i} if the hidden neuron i excludes patterns belonging to class j and zero otherwise.

Let $\mathbf{W}_l^h$ be the weights between the lth hidden neuron and inputs. Let $\mathbf{W}_m^o$ be the weights between the output neuron for class m and hidden neurons, and W_{ml}^o be the weight between the output neuron for class m and the lth hidden neuron, respectively. The pseudo-code shown in Figure 5.2 summarizes the process of network construction.

5.2.3 Use of Network in Classification

The outputs in the output layer are computed by the *winner-take-all (WTA)* strategy. The output neuron m that has the highest net input produces 1 and all the other neurons produce 0's. The WTA strategy and the weight setting explained in section 5.2.2 guarantee 100%

Initialize the number of hidden neurons: $h = 0$;
while $S \neq \phi$
do

1. Double all existing weights (if any) between hidden and output neurons:
   $\mathbf{W}_m^o = \mathbf{W}_m^o * 2 \ \forall m \in 1, \cdots, h$
2. Increment the number of hidden neurons: $h = h + 1$
3. Inter-pattern distance based:
   Identify a row k of $\mathcal{D}$ that excludes the largest subset of patterns in S that belong to the same class m as follows:
   (a) For each row $r = 1, \cdots, N$ do
      i. Let i_r and j_r be column indices (corresponding to row r) for the matrix $\mathcal{D}$ such that the patterns corresponding to the elements $\mathcal{D}[r, i_r]$, $\mathcal{D}[r, i_r + 1]$, ... $\mathcal{D}[r, j_r]$ all belong to the same class and also belong to S.
      ii. Let $c_r = j_r - i_r + 1$ (the number of patterns excluded).
   (b) Select k to be the one for which the corresponding c_k is the largest:
   $k = \arg\max_r c_r$
   (c) Let S_k be the corresponding set of patterns that are excluded by pattern $\mathbf{X}^k$, $d_{low}^k = \mathcal{D}[k, i_k]$ (distance to the closest pattern of the cluster) and $d_{high}^k = \mathcal{D}[k, j_k]$ (distance to the farthest pattern of the cluster).
4. Attribute based:
   Analogously, using $\mathcal{D}'$ identify an attribute a that excludes the largest number of patterns in S that belong to the same output class m (i.e.. identify a for which c_a is the largest among all attributes.); Let S_a be the corresponding set of patterns from S that are excluded by attribute a, d_{low}^a and d_{low}^a be the minimum and maximum values respectively for attribute a among the patterns in set S_a.
5. **if** [Inter-pattern distance based] **then**
   (a) Define a spherical threshold neuron with $\mathbf{W}^h = \mathbf{X}^k, \theta_{low} = d_{low}^k, \theta_{high} = d_{high}^k$.
   (b) $S = S - S_k$

   else

   (a) Define a neuron corresponding to attribute a with $\theta_{low} = d_{low}^a, \theta_{high} = d_{high}^a$.
   (b) $S = S - S_a$

   endif
6. Connect the new hidden neuron to output neurons: $W_{mh}^o = 1; W_{nh}^o = 0 \ \forall n \neq m$

end while

Figure 5.2 DistAI algorithm.

training accuracy for any finite non-contradictory set of training patterns. (See section 5.2.5 for the detailed convergence proof.)

The generalization accuracy of a test set is computed in the same way. Each test pattern is fed into the network and the outputs are computed by the WTA strategy. If there are one or more hidden neurons that produce 1 (i.e., there exist one or more hidden neurons that include the test pattern within their thresholds), the outputs are computed by the WTA strategy in the output layer. Otherwise (i.e., all hidden neurons produce 0's and all output neurons produce 0's as well), the distance between the test pattern and the thresholds of each hidden neuron is computed. The hidden neuron that has the minimum distance is chosen to produce 1. Then the outputs are computed again in the output layer to compare with the desired classification.

5.2.4 Example

We illustrate the operation of DistAI using the simple XOR problem. We will assume the use of Manhattan distance metric. There are four training patterns ($S = \{\mathbf{X}^1, \mathbf{X}^2, \mathbf{X}^3, \mathbf{X}^4\}$):

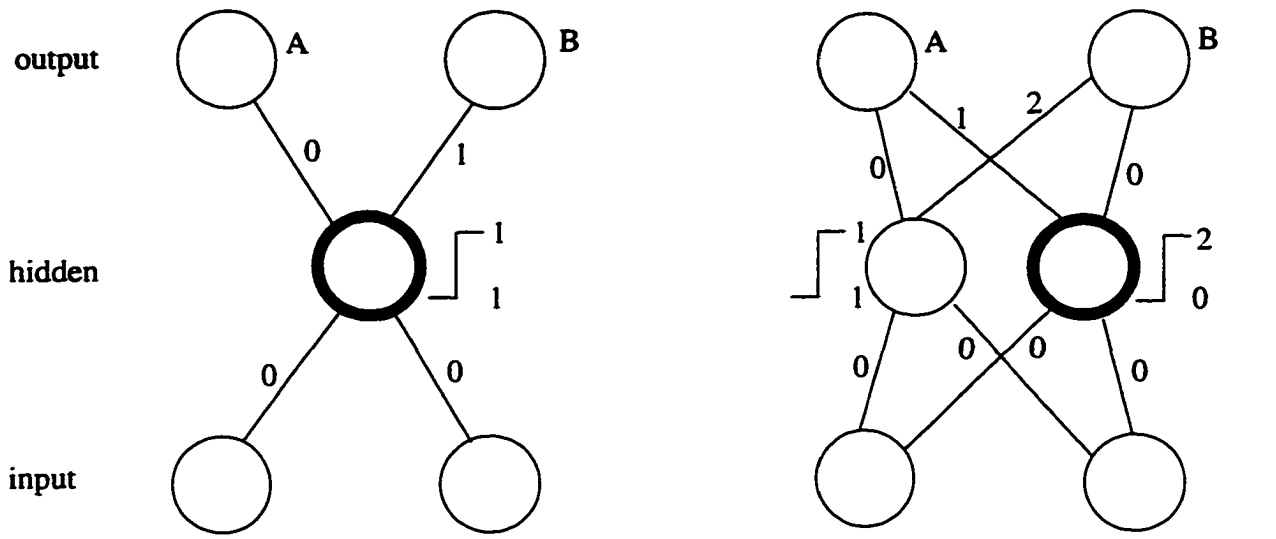
input			class
$\mathbf{X}^1$:	0	0	A
$\mathbf{X}^2$:	0	1	B
$\mathbf{X}^3$:	1	0	B
$\mathbf{X}^4$:	1	1	A

This yields the following distance matrix after sorted:

$$\mathcal{D} = \begin{pmatrix} 0\,1\,1\,2 \\ 0\,1\,1\,2 \\ 0\,1\,1\,2 \\ 0\,1\,1\,2 \end{pmatrix}$$

The first row of the matrix is the distance of $\mathbf{X}^1, \mathbf{X}^2, \mathbf{X}^3$ and $\mathbf{X}^4$ from pattern $\mathbf{X}^1$. The second row of the matrix is the distance of $\mathbf{X}^2, \mathbf{X}^1, \mathbf{X}^4$ and $\mathbf{X}^3$ from $\mathbf{X}^2$. The third row of the matrix is the distance of $\mathbf{X}^3, \mathbf{X}^1, \mathbf{X}^4$ and $\mathbf{X}^2$ from $\mathbf{X}^3$. The last row of the matrix is the distance of $\mathbf{X}^4, \mathbf{X}^2, \mathbf{X}^3$ and $\mathbf{X}^1$ from $\mathbf{X}^4$.

$\mathbf{X}^1$ excludes the maximum number of patterns from a single class (i.e., $S_k = \{\mathbf{X}^2, \mathbf{X}^3\}$, class = B). A hidden neuron is introduced for this cluster with $\mathbf{W}_1^h = [0\ \ 0], \theta_{low} = \theta_{high} = 1, W_{B1}^o = 1, W_{A1}^o = 0$. $\mathbf{X}^2$ and $\mathbf{X}^3$ are now eliminated from further consideration (i.e., $S = S - S_k = \{\mathbf{X}^1, \mathbf{X}^4\}$) The remaining patterns ($S_k = \{\mathbf{X}^1, \mathbf{X}^4\}$, class = A) can be excluded by any pattern (say, $\mathbf{X}^1$ again) with another hidden neuron with $\mathbf{W}_2^h = [0\ \ 0]. \theta_{low} = 0, \theta_{high} = 2, W_{A2}^o = 1, W_{B2}^o = 0, W_{A1}^o = W_{A1}^o * 2 = 0, W_{B1}^o = W_{B1}^o * 2 = 2$. Now the algorithm stops since the entire training set is correctly classified (i.e., $S = S - S_k = \phi$). Figure 5.3 shows the network construction process.



(a) after the first neuron is introduced

(b) after the second neuron is introduced (final network)

Figure 5.3 Process of network construction for the example in DistAl.

5.2.5 Convergence Proof

Theorem:

Given a finite non-contradictory set of training examples E, DistAl is guaranteed to converge to zero classification error after adding a finite number (h) of hidden neurons, where $h \leq |E|$. (In practice, $h \ll |E|$).

Proof:

Let Z_i be the set of patterns that are excluded by ith hidden neuron. Each hidden neuron finds the largest subset of patterns to be excluded. DistAI keeps introducing a hidden neuron until S becomes an empty set (i.e., $S = S - Z_i$). Since $S = \{\mathbf{X}^1, \cdots, \mathbf{X}^N\}$ is the training set with the cardinality of N, $h = |Z_1, Z_2, \cdots, Z_h| \leq N$ where Z_h is the last subset of patterns to be eliminated. It is clear that at least one pattern ($\mathbf{X}^p$) can be excluded by a new hidden neuron i with $\mathbf{W}_i^h = \mathbf{X}^p$ and 0 thresholds. [1] Since there are a finite number of patterns in the training set, and since each added hidden neuron is guaranteed to correctly classify a non-empty subset of the training set which is then eliminated from further consideration, no more than $|E|$ hidden neurons are needed.

The internal representation of the hidden layer for a pattern $\mathbf{X}^p$ (which is a member of the ith cluster) has the form

$$\mathbf{H}^p = (0, 0, \cdots, 0, 1, *, \cdots, *) \tag{5.1}$$

(it has 0's in the first $i-1$ hidden neurons, 1 in the ith hidden neuron and either 0 or 1 in the remaining hidden neurons) for a network with h hidden neurons. The weights from hidden to output neurons are set directly as explained in Section 5.2.2:

$$W_{ji}^o = \begin{cases} 2^{h-i} \text{ if } j \text{ is the right class of hidden neuron } i \\ 0 \text{ otherwise} \end{cases}$$

Consider a pattern $\mathbf{X}^p$ which belongs to the subset Z_i of patterns excluded by the ith hidden neuron that represents the pattern $\mathbf{X}^k$. Let c_j be the classification of $\mathbf{X}^k$. Then $W_{ji}^o > W_{li}^o \ \forall j \neq l$. Also, the internal representation (5.1) guarantees the net input of output neuron j to be larger than that of any other output neuron. Consequently, $\mathbf{X}^p$ is correctly classified in the output layer by the WTA strategy. As an example, assume $\mathbf{H}^p = (1, 1, 1)$ for a pattern $\mathbf{X}^p$ belonging to class A, and the hidden neurons represent clusters for class A, B and B, respectively. Then, when $\mathbf{X}^p$ is fed into input neurons, the net input to the output neuron for

[1] Note that this is not always true for maximum value distance metric and attribute-based approach. That is because there can be many patterns of different classifications that have the same maximum values/attributes values. Therefore, the convergence proof given here and the complexity analysis in Section 5.2.6 apply to distance-based approaches (excluding Maximum value metric), but not attribute-based approach.

class A will be $2^{3-1} = 4$ and that to the output neuron for class B will be $2^{3-2} + 2^{3-3} = 3$. Thus, $\mathbf{X}^p$ will be correctly classified as class A.

Therefore, DistAI is guaranteed to converge to zero classification error after adding a finite number of hidden neurons for a finite non-contradictory set of training examples. □

5.2.6 Complexity Analysis

This section presents the complexity analysis for DistAI. The complexity analysis assumes that network construction is based on a single distance metric.

Let N_{pat} be the number of training patterns and N_{att} be the number of attributes in a dataset, respectively. Let N_{out} be the number of output neurons. Assume $N_{pat} > N_{att}$ and $N_{pat} \gg max[N_{out}, h]$.

1. *Time Complexity*

   Computing and sorting the distance matrix $\mathcal{D}$ takes $\mathcal{O}(max[N_{pat}^2 \cdot N_{att}, N_{pat}^2 \cdot \log N_{pat}])$. [2] Now, consider the pseudo-code given in Section 5.2.2. Step 1 takes $\mathcal{O}(N_{out} \cdot h)$. Step 2 takes $\mathcal{O}(1)$. Step 3 takes $\mathcal{O}(N_{pat}^2)$ because we need to go through the entire matrix $\mathcal{D}$ to determine S_k. [3] Step 5 takes $\mathcal{O}(N_{pat})$ to update S. Step 6 takes $\mathcal{O}(N_{out})$. Thus, the **while** loop takes $\mathcal{O}(N_{pat}^3)$ in the worst case. Therefore, the overall worst-case time complexity is $\mathcal{O}(N_{pat}^3)$. In practice, DistAI runs significantly faster than the worst-case time complexity because it eliminates a subset of elements from the original training set instead of a single pattern. This makes DistAI particularly well-suited for largescale data mining tasks.

2. *Space Complexity*

   The space requirement for the input patterns and their targets is $\mathcal{O}(N_{pat} \cdot [N_{att} + N_{out}])$. The weights require $\mathcal{O}(N_{out} \cdot h + h \cdot N_{in})$. The distance matrix requires $\mathcal{O}(N_{pat}^2)$. Thus, the total space complexity is $\mathcal{O}(N_{pat}^2)$.

[2] Computation of $\mathcal{D}'$ in attribute-based approach takes only $\mathcal{O}(N_{att} \cdot N_{pat} \log N_{pat})$ because distance computation is not necessary.

[3] Step 4 is not considered here because it is used only with the attribute-based metric. The time required for step 4 is comparable to the time required for step 3.

5.3 Experimental Evaluation of DistAI

This section presents results of experiments using DistAI on several benchmark problems and compares them with the results of other learning algorithms.

5.3.1 Datasets

Two artificial datasets (parity and two spirals) and a wide range of real-world datasets from the machine learning data repository at the University of California at Irvine [Murphy & Aha, 1994] were chosen to test the performance of DistAI. Table 5.1 summarizes the characteristics of the datasets selected for our experiments. Size is the number of patterns in the dataset, Dimension is the number of input attributes, Missing? is whether there are any missing values, and Class is the number of output classes in the table.

5.3.2 Experimental Results

DistAI is deterministic in the sense that its behavior is always identical for a given training set. Most other constructive learning algorithms are non-deterministic because their behavior is not always identical in different runs with the same training set and even with the same learning parameters due to the randomness in selecting initial weights, pattern presentations, and so on. Therefore, just one run of DistAI per dataset is sufficient to study the performance.

5.3.2.1 Parity Datasets

The seven, eight and nine-bit parity datasets (**P7, P8, P9**) were used to evaluate the performance of DistAI in terms of the network size. The Manhattan distance metric was used to train the entire set of patterns. Table 5.2 presents the size of the network generated by several algorithms. (A '-' indicate that the result is not reported in the corresponding reference). It shows that DistAI is capable of generating compact networks comparable to other algorithms for non-trivial tasks like the parity problem. Note that DistAI is also fast. Since DistAI does not require iterative perceptron training procedure and keeps eliminating a subset of patterns

Table 5.1 Datasets used in the experiments with DistAl.

Dataset	Size	Dimension	Attribute Type	Missing?	Class
7-bit parity (**P7**)	128	7	numeric	No	2
8-bit parity (**P8**)	256	8	numeric	No	2
9-bit parity (**P9**)	512	9	numeric	No	2
two spirals (**2SP**)	192	2	numeric	No	2
annealing database (**Annealing**)	798	38	numeric, nominal	Yes	5
audiology database (**Audiology**)	200	69	nominal	Yes	24
pittsburgh bridges (**Bridges**)	105	11	numeric, nominal	Yes	6
breast cancer (**Cancer**)	699	9	numeric	Yes	2
credit screening (**CRX**)	690	15	numeric, nominal	Yes	2
flag database (**Flag**)	194	28	numeric, nominal	No	8
glass identification (**Glass**)	214	9	numeric	No	6
heart disease (**Heart**)	270	13	numeric, nominal	No	2
heart disease [Cleveland](**HeartCle**)	303	13	numeric, nominal	Yes	2
heart disease [Hungarian](**HeartHun**)	294	13	numeric, nominal	Yes	2
heart disease [Long Beach](**HeartLB**)	200	13	numeric, nominal	Yes	2
heart disease [Swiss](**HeartSwi**)	123	13	numeric, nominal	Yes	2
hepatitis domain (**Hepatitis**)	155	19	numeric, nominal	Yes	2
horse colic (**Horse**)	300	22	numeric, nominal	Yes	2
ionosphere structure (**Ionosphere**)	351	34	numeric	No	2
iris plants (**Iris**)	150	4	numeric	No	3
liver disorders (**Liver**)	345	6	numeric	No	2
monks problems (**Monks-1,2,3**)	432	6	nominal	No	2
pima indians diabetes (**Pima**)	768	8	numeric	No	2
DNA sequences (**Promoters**)	106	57	nominal	No	2
sonar classification (**Sonar**)	208	60	numeric	No	2
large soybean (**Soylarge**)	307	35	nominal	Yes	19
small soybean (**Soysmall**)	47	35	nominal	No	4
vehicle silhouettes (**Vehicle**)	846	18	numeric	No	4
house votes (**Votes**)	435	16	nominal	Yes	2
vowel recognition (**Vowel**)	528	10	numeric	No	11
wine recognition (**Wine**)	178	13	numeric	No	3
zoo database (**Zoo**)	101	16	numeric, nominal	No	7

Table 5.2 Network size for parity datasets.

Algorithm	P7	P8	P9
DistAI	5	5	6
GA-MLP [Andersen & Tsoi, 1993]	9	15	-
Perceptron cascade [Burgess, 1994]	3	4	4
Cascade correlation [Fahlman & Lebiere, 1990]	4-5	5-6	-
Upstart [Frean, 1990]	6	7	8
Growth algorithm [Golea & Marchand, 1990]	7	8	9
Sequential [Marchand *et al.*, 1990]	7	8	9
Tiling [Mézard & Nadal, 1989]	7	8	9
Tower [Nadal, 1989; Gallant, 1990]	3.5	4	4.5

that are not considered further in the learning process, it converges significantly fast. [4]

5.3.2.2 Various Datasets from UCI Repository

DistAI was run once for each distance metric to compare the performance in terms of the generalization accuracy and the network size. A simple pruning technique was implemented to produce compact networks: When a new hidden neuron is introduced, the generalization accuracy of the network is computed. The current best generalization accuracy is stored in a *pocket* along with the network size. After the training is completed (i.e., 100% training accuracy is obtained) or no further training is possible (i.e., the limit of allowable hidden neurons (currently set to 100) is reached or no more patterns can be eliminated in Maximum value metric or attribute-based approach), the network with the best generalization accuracy in the pocket is restored by pruning the unnecessary hidden neurons.

A 10-fold cross-validation was performed for each dataset with all the distance metrics introduced in Section 5.2.1 and its performance was shown in Tables 5.3-5.8. Different methods of normalization are indicated by either r (range-based) or s (standard deviation-based), and different methods for handling nominal values are indicated by either v (value difference) or

[4] It is not feasible to make a fair, thorough comparison of speeds of different algorithms. DistAI converged fairly quickly for almost all datasets. (See Section 5.2.6 for detailed analysis of time complexity). GA-MLP [Andersen & Tsoi, 1993] is based on a genetic algorithm and thus it usually takes significant amount of time to get a quality solution. Cascade correlation [Fahlman & Lebiere, 1990] uses *Quickprop* [Fahlman, 1988]. *Quickprop* uses an iterative gradient descent method based on a second order heuristic.

Table 5.3 Results of Range, Value-difference based distance metrics.

	Euclidean [r,v]		Manhattan [r,v]		Maximum value [r,v]	
Dataset	*Accuracy*	*Hidden*	*Accuracy*	*Hidden*	*Accuracy*	*Hidden*
2SP	79.5 ± 10.1	8.7 ± 1.5	72.1 ± 5.8	8.5 ± 3.7	70.5 ± 8.9	10.1 ± 3.7
Annealing	**96.6** ± 2.0	12.1 ± 2.4	93.3 ± 2.8	15.5 ± 3.4	0.0 ± 0.0	0.0 ± 0.0
Audiology	**66.0** ± 9.7	24.7 ± 4.8	59.0 ± 8.6	26.7 ± 3.0	1.0 ± 2.0	0.6 ± 1.2
Bridge	55.0 ± 15.6	3.5 ± 2.1	59.0 ± 10.4	3.2 ± 2.6	45.0 ± 16.9	3.4 ± 1.5
Cancer	**97.8** ± 1.2	2.9 ± 1.2	97.5 ± 1.7	3.9 ± 1.4	95.1 ± 1.5	6.2 ± 5.1
CRX	**87.7** ± 3.3	7.7 ± 6.9	87.5 ± 3.8	7.3 ± 4.3	86.4 ± 3.3	7.7 ± 3.5
Flag	63.7 ± 8.0	5.7 ± 3.2	64.7 ± 11.3	6.0 ± 3.6	57.9 ± 5.8	7.0 ± 2.7
Glass	**70.5** ± 8.5	9.8 ± 6.9	66.2 ± 4.5	9.9 ± 6.5	67.6 ± 7.6	10.1 ± 5.6
Heart	83.7 ± 5.3	3.3 ± 1.8	84.8 ± 4.8	5.7 ± 3.6	85.2 ± 3.3	7.6 ± 4.5
HeartCle	**85.3** ± 7.2	4.6 ± 3.8	**85.3** ± 3.4	6.0 ± 2.9	82.3 ± 4.5	10.9 ± 6.7
HeartHun	84.5 ± 5.8	6.7 ± 2.8	84.8 ± 5.6	6.6 ± 2.9	0.0 ± 0.0	0.0 ± 0.0
HeartLB	78.5 ± 9.2	5.0 ± 3.5	77.5 ± 6.8	4.9 ± 3.0	0.0 ± 0.0	0.0 ± 0.0
HeartSwi	93.3 ± 3.3	2.0 ± 0.0	93.3 ± 5.0	2.2 ± 0.8	0.0 ± 0.0	0.0 ± 0.0
Hepatitis	83.3 ± 4.5	3.0 ± 1.3	83.3 ± 6.2	2.5 ± 0.8	79.3 ± 8.7	2.0 ± 0.0
Horse	**86.0** ± 3.6	5.3 ± 4.5	84.7 ± 4.3	5.1 ± 3.2	63.7 ± 5.9	2.0 ± 0.0
Ionosphere	93.1 ± 4.5	6.8 ± 1.4	90.0 ± 5.8	5.8 ± 2.1	91.4 ± 4.2	5.5 ± 1.7
Iris	93.1 ± 4.5	6.8 ± 1.4	90.0 ± 5.8	5.8 ± 2.1	91.4 ± 4.2	5.5 ± 1.7
Liver	67.7 ± 6.8	7.8 ± 4.5	63.5 ± 8.2	6.4 ± 6.8	67.4 ± 5.3	7.1 ± 3.7
Monks-1	90.0 ± 9.3	7.4 ± 4.4	89.1 ± 7.7	7.4 ± 5.1	82.8 ± 9.1	9.6 ± 4.9
Monks-2	79.8 ± 10.4	8.4 ± 4.5	79.5 ± 10.4	13.0 ± 9.5	82.8 ± 9.4	6.4 ± 2.9
Monks-3	**99.1** ± 1.5	3.0 ± 0.0	98.6 ± 1.9	3.0 ± 0.6	98.6 ± 1.9	2.1 ± 0.3
Pima	74.3 ± 3.2	9.5 ± 6.7	73.4 ± 4.0	13.2 ± 7.8	73.7 ± 5.2	8.3 ± 5.0
Promoters	87.0 ± 11.0	2.8 ± 0.4	**88.0** ± 7.5	2.2 ± 0.4	85.0 ± 8.1	2.8 ± 0.6
Sonar	**83.0** ± 7.8	6.4 ± 2.7	81.5 ± 9.5	4.8 ± 2.4	78.5 ± 8.1	7.5 ± 3.8
Soylarge	**81.0** ± 5.6	20.2 ± 3.2	74.3 ± 9.3	21.6 ± 5.0	67.7 ± 4.5	16.7 ± 2.4
Soysmall	90.0 ± 16.6	3.4 ± 0.5	92.5 ± 16.0	3.6 ± 0.5	**97.5** ± 7.5	3.6 ± 0.5
Vehicle	64.1 ± 6.5	29.5 ± 13.3	61.7 ± 3.2	25.9 ± 18.3	57.0 ± 4.7	49.4 ± 22.2
Votes	**96.1** ± 1.5	3.2 ± 1.5	95.4 ± 2.3	3.7 ± 1.2	78.8 ± 8.1	3.6 ± 1.4
Vowel	65.2 ± 6.9	34.6 ± 8.5	65.8 ± 6.4	40.7 ± 8.5	61.7 ± 8.3	39.5 ± 7.7
Wine	92.9 ± 5.8	4.3 ± 0.8	92.9 ± 5.8	4.1 ± 0.7	94.1 ± 6.4	4.7 ± 0.6
Zoo	**96.0** ± 4.9	6.1 ± 1.1	**96.0** ± 8.0	6.1 ± 0.9	93.9 ± 4.6	6.0 ± 1.2

Table 5.4 Results of Standard Deviation, Value-difference based distance metrics.

	Euclidean [s,v]		Manhattan [s,v]		Maximum value [s,v]	
Dataset	*Accuracy*	*Hidden*	*Accuracy*	*Hidden*	*Accuracy*	*Hidden*
2SP	**83.7** ± 7.6	7.7 ± 1.8	69.5 ± 6.1	7.1 ± 3.9	72.1 ± 6.7	8.8 ± 3.1
Annealing	96.3 ± 1.4	10.6 ± 2.8	93.9 ± 2.3	13.7 ± 3.4	0.0 ± 0.0	0.0 ± 0.0
Audiology	**66.0** ± 9.7	24.7 ± 4.8	59.0 ± 8.6	26.7 ± 3.0	1.0 ± 2.0	0.6 ± 1.2
Bridge	56.0 ± 17.4	4.0 ± 3.6	59.0 ± 13.0	3.4 ± 2.7	52.0 ± 14.7	4.0 ± 2.8
Cancer	96.8 ± 2.0	4.0 ± 1.6	96.8 ± 1.9	4.5 ± 2.6	95.4 ± 1.7	10.4 ± 4.4
CRX	87.4 ± 3.6	7.2 ± 3.7	87.0 ± 4.1	7.0 ± 4.5	86.4 ± 4.5	6.1 ± 4.6
Flag	60.5 ± 8.2	6.4 ± 4.6	**65.8** ± 9.5	9.1 ± 6.2	55.3 ± 10.3	11.1 ± 9.2
Glass	68.1 ± 7.7	11.5 ± 7.7	66.2 ± 5.8	7.3 ± 3.6	69.5 ± 6.8	9.5 ± 7.7
Heart	82.6 ± 5.0	3.6 ± 1.6	85.6 ± 5.1	4.7 ± 3.1	81.5 ± 6.2	7.7 ± 5.9
HeartCle	81.7 ± 4.8	3.9 ± 2.7	83.7 ± 4.6	4.6 ± 3.3	83.7 ± 6.7	5.6 ± 4.5
HeartHun	84.8 ± 6.4	7.0 ± 4.1	83.1 ± 4.7	5.5 ± 3.4	0.0 ± 0.0	0.0 ± 0.0
HeartLB	76.5 ± 8.7	3.3 ± 2.8	77.5 ± 6.8	3.9 ± 2.5	0.0 ± 0.0	0.0 ± 0.0
HeartSwi	**94.2** ± 3.8	2.2 ± 0.6	**94.2** ± 3.8	2.3 ± 0.9	0.0 ± 0.0	0.0 ± 0.0
Hepatitis	**84.7** ± 9.5	6.2 ± 4.0	84.0 ± 6.8	4.6 ± 3.1	**84.7** ± 9.5	4.2 ± 1.8
Horse	83.0 ± 5.7	4.4 ± 3.5	83.7 ± 6.9	7.9 ± 4.4	63.7 ± 5.9	2.0 ± 0.0
Ionosphere	92.9 ± 5.5	6.9 ± 2.1	91.4 ± 5.9	5.8 ± 1.4	92.6 ± 4.1	5.3 ± 1.9
Iris	92.9 ± 5.5	6.9 ± 2.1	91.3 ± 6.0	3.3 ± 1.0	94.0 ± 6.3	4.0 ± 1.2
Liver	66.5 ± 5.1	9.9 ± 5.7	64.4 ± 4.6	10.6 ± 8.3	66.2 ± 6.2	11.3 ± 7.6
Monks-1	90.0 ± 9.3	7.4 ± 4.4	89.1 ± 7.7	7.4 ± 5.1	82.8 ± 9.1	9.6 ± 4.9
Monks-2	79.8 ± 10.4	8.4 ± 4.5	79.5 ± 10.4	13.0 ± 9.5	82.8 ± 9.4	6.4 ± 2.9
Monks-3	**99.1** ± 1.5	3.0 ± 0.0	98.6 ± 1.9	3.0 ± 0.6	98.6 ± 1.9	2.1 ± 0.3
Pima	74.2 ± 3.9	10.0 ± 3.8	**76.3** ± 5.1	8.1 ± 4.9	74.7 ± 4.0	13.1 ± 10.8
Promoters	87.0 ± 11.0	2.8 ± 0.4	**88.0** ± 7.5	2.2 ± 0.4	85.0 ± 8.1	2.8 ± 0.6
Sonar	73.5 ± 7.4	4.6 ± 3.1	78.5 ± 8.1	4.8 ± 2.1	73.5 ± 7.4	5.6 ± 3.6
Soylarge	**81.0** ± 5.6	20.2 ± 3.2	74.3 ± 9.3	21.6 ± 5.0	67.7 ± 4.5	16.7 ± 2.4
Soysmall	90.0 ± 16.6	3.4 ± 0.5	92.5 ± 16.0	3.6 ± 0.5	**97.5** ± 7.5	3.6 ± 0.5
Vehicle	65.1 ± 4.0	24.4 ± 9.6	**65.4** ± 3.5	23.7 ± 5.0	62.1 ± 4.7	52.9 ± 18.6
Votes	**96.1** ± 1.5	3.2 ± 1.5	95.4 ± 2.3	3.7 ± 1.2	78.8 ± 8.1	3.6 ± 1.4
Vowel	66.7 ± 7.5	31.2 ± 10.1	65.0 ± 7.7	36.3 ± 8.5	57.9 ± 8.8	39.2 ± 13.5
Wine	95.9 ± 4.6	4.9 ± 0.3	92.4 ± 7.5	4.4 ± 0.7	92.9 ± 5.8	4.4 ± 0.9
Zoo	**96.0** ± 4.9	6.1 ± 1.1	**96.0** ± 8.0	6.1 ± 0.9	93.0 ± 4.6	6.0 ± 1.2

Table 5.5 Results of Range, Overlap based distance metrics.

	Euclidean [r,o]		Manhattan [r,o]		Maximum value [r,o]	
Dataset	*Accuracy*	*Hidden*	*Accuracy*	*Hidden*	*Accuracy*	*Hidden*
2SP	79.5 ± 10.1	8.7 ± 1.5	72.1 ± 5.8	8.5 ± 3.7	70.5 ± 8.9	10.1 ± 3.7
Annealing	94.7 ± 1.8	14.6 ± 3.6	93.2 ± 2.5	15.6 ± 5.0	0.0 ± 0.0	0.0 ± 0.0
Audiology	**66.0** ± 10.0	27.3 ± 7.4	**66.0** ± 10.0	27.3 ± 7.4	1.0 ± 2.0	0.6 ± 1.2
Bridge	60.0 ± 10.0	4.8 ± 3.0	60.0 ± 10.0	6.3 ± 5.0	36.0 ± 18.0	10.0 ± 17.1
Cancer	**97.8** ± 1.2	2.9 ± 1.2	97.5 ± 1.7	3.9 ± 1.4	95.1 ± 1.5	6.2 ± 5.1
CRX	83.8 ± 5.3	9.7 ± 5.1	83.9 ± 3.8	9.1 ± 4.4	61.9 ± 7.9	56.4 ± 20.7
Flag	47.4 ± 7.1	6.9 ± 5.4	50.5 ± 10.8	10.3 ± 6.4	24.7 ± 9.4	2.1 ± 0.3
Glass	**70.5** ± 8.5	9.8 ± 6.9	66.2 ± 4.5	9.9 ± 6.5	67.6 ± 7.6	10.1 ± 5.6
Heart	**86.7** ± 7.6	5.7 ± 4.4	86.3 ± 5.8	4.1 ± 3.0	73.3 ± 4.9	26.5 ± 25.0
HeartCle	83.0 ± 5.5	4.9 ± 2.7	**85.3** ± 2.7	3.4 ± 1.1	71.7 ± 10.3	23.6 ± 15.3
HeartHun	**85.9** ± 6.3	5.0 ± 2.9	84.8 ± 3.8	4.5 ± 3.0	0.0 ± 0.0	0.0 ± 0.0
HeartLB	77.0 ± 9.8	3.4 ± 2.5	**80.0** ± 7.4	5.1 ± 2.6	0.0 ± 0.0	0.0 ± 0.0
HeartSwi	**94.2** ± 3.8	2.3 ± 0.9	**94.2** ± 3.8	2.3 ± 0.9	0.0 ± 0.0	0.0 ± 0.0
Hepatitis	83.3 ± 4.5	3.0 ± 1.3	83.3 ± 6.2	2.5 ± 0.8	79.3 ± 8.7	2.0 ± 0.0
Horse	84.0 ± 6.3	5.6 ± 2.2	85.7 ± 7.9	4.1 ± 2.5	63.7 ± 5.9	2.0 ± 0.0
Ionosphere	93.1 ± 4.5	6.8 ± 1.4	90.0 ± 5.8	5.8 ± 2.1	91.4 ± 4.2	5.5 ± 1.7
Iris	96.0 ± 4.4	3.4 ± 0.7	96.0 ± 3.3	3.4 ± 0.7	91.3 ± 6.7	3.3 ± 0.5
Liver	67.7 ± 6.8	7.8 ± 4.5	63.5 ± 8.2	6.4 ± 6.8	67.4 ± 5.3	7.1 ± 3.7
Monks-1	**90.9** ± 7.1	26.9 ± 7.5	**90.9** ± 7.1	26.9 ± 7.5	49.3 ± 7.1	2.0 ± 0.0
Monks-2	**100** ± 0.0	2.7 ± 2.1	**100** ± 0.0	2.7 ± 2.1	33.0 ± 4.3	2.0 ± 0.0
Monks-3	91.6 ± 4.4	16.2 ± 4.4	91.6 ± 4.4	16.2 ± 4.4	49.3 ± 6.6	2.0 ± 0.0
Pima	74.3 ± 3.2	9.5 ± 6.7	73.4 ± 4.0	13.2 ± 7.8	73.7 ± 5.2	8.3 ± 5.0
Promoters	83.0 ± 6.4	3.4 ± 1.4	83.0 ± 6.4	3.4 ± 1.4	56.0 ± 6.6	20.0 ± 36.0
Sonar	**83.0** ± 7.8	6.4 ± 2.7	81.5 ± 9.5	4.8 ± 2.4	76.0 ± 9.2	7.5 ± 3.8
Soylarge	75.0 ± 5.2	26.3 ± 4.7	75.0 ± 5.2	26.3 ± 4.7	12.3 ± 6.8	2.0 ± 0.0
Soysmall	**97.5** ± 7.5	3.9 ± 0.3	**97.5** ± 7.5	0.9 ± 0.3	30.0 ± 21.8	13.3 ± 18.2
Vehicle	64.1 ± 6.5	29.5 ± 13.3	61.7 ± 3.2	25.9 ± 18.3	57.0 ± 4.7	49.4 ± 22.2
Votes	95.6 ± 2.6	6.1 ± 2.3	95.6 ± 2.6	6.1 ± 2.3	47.0 ± 8.1	42.5 ± 29.2
Vowel	65.2 ± 6.9	34.6 ± 8.5	65.8 ± 6.4	40.7 ± 8.5	61.7 ± 8.3	39.5 ± 7.7
Wine	92.9 ± 5.8	4.3 ± 0.8	92.9 ± 5.8	4.1 ± 0.7	94.1 ± 6.4	4.7 ± 0.6
Zoo	92.0 ± 7.5	6.2 ± 0.9	92.0 ± 7.5	6.2 ± 0.9	75.0 ± 12.9	33.4 ± 17.4

Table 5.6 Results of Standard Deviation, Overlap based distance metrics.

	Euclidean [s,o]		Manhattan [s,o]		Maximum value [s,o]	
Dataset	*Accuracy*	*Hidden*	*Accuracy*	*Hidden*	*Accuracy*	*Hidden*
2SP	**83.7** ± 7.6	7.7 ± 1.8	69.5 ± 6.1	7.1 ± 3.9	72.1 ± 6.7	8.8 ± 3.1
Annealing	95.2 ± 1.9	14.5 ± 2.4	94.8 ± 3.0	17.5 ± 2.5	0.0 ± 0.0	0.0 ± 0.0
Audiology	**66.0** ± 10.0	27.3 ± 7.4	**66.0** ± 10.0	27.3 ± 7.4	1.0 ± 2.0	0.6 ± 1.2
Bridge	**63.0** ± 7.8	5.2 ± 3.3	60.0 ± 7.8	4.2 ± 2.7	38.0 ± 14.7	15.3 ± 21.9
Cancer	96.8 ± 2.0	4.0 ± 1.6	96.8 ± 1.9	4.5 ± 2.6	95.4 ± 1.7	10.4 ± 4.4
CRX	85.2 ± 5.6	10.5 ± 5.4	84.9 ± 6.3	9.1 ± 6.3	58.7 ± 6.5	44.0 ± 24.0
Flag	46.8 ± 7.2	7.2 ± 4.4	51.1 ± 8.8	9.1 ± 8.2	31.6 ± 10.8	3.6 ± 1.4
Glass	68.1 ± 7.7	11.5 ± 7.7	66.2 ± 5.8	7.3 ± 3.6	69.5 ± 6.8	9.5 ± 7.7
Heart	85.9 ± 6.4	5.5 ± 3.3	85.6 ± 4.5	5.2 ± 3.6	71.1 ± 5.2	26.4 ± 17.0
HeartCle	82.0 ± 4.5	3.9 ± 2.4	83.3 ± 7.0	5.4 ± 4.2	67.0 ± 7.1	26.7 ± 17.5
HeartHun	82.1 ± 4.8	5.4 ± 4.4	85.5 ± 4.6	5.2 ± 3.7	0.0 ± 0.0	0.0 ± 0.0
HeartLB	77.0 ± 7.5	4.3 ± 3.4	79.0 ± 6.2	5.9 ± 4.4	0.0 ± 0.0	0.0 ± 0.0
HeartSwi	**94.2** ± 3.8	2.2 ± 0.6	**94.2** ± 3.8	2.2 ± 0.6	0.0 ± 0.0	0.0 ± 0.0
Hepatitis	**84.7** ± 9.5	6.2 ± 4.0	84.0 ± 6.8	4.6 ± 3.1	**84.7** ± 9.5	4.2 ± 1.8
Horse	80.0 ± 5.2	10.1 ± 6.2	84.3 ± 4.2	4.5 ± 2.0	63.7 ± 5.9	2.0 ± 0.0
Ionosphere	92.9 ± 5.5	6.9 ± 2.1	91.4 ± 5.9	5.8 ± 1.4	92.6 ± 4.1	5.3 ± 1.9
Iris	94.0 ± 3.6	3.8 ± 1.2	91.3 ± 6.0	3.3 ± 1.0	94.0 ± 6.3	4.0 ± 1.2
Liver	66.5 ± 5.1	9.9 ± 5.7	64.4 ± 4.6	10.6 ± 8.3	66.2 ± 6.2	11.3 ± 7.6
Monks-1	**90.9** ± 7.1	26.9 ± 7.5	**90.9** ± 7.1	26.9 ± 7.5	49.3 ± 7.1	2.0 ± 0.0
Monks-2	**100** ± 0.0	2.7 ± 2.1	**100** ± 0.0	2.7 ± 2.1	33.1 ± 0.5	2.0 ± 0.0
Monks-3	91.6 ± 4.4	16.2 ± 4.4	91.6 ± 4.4	16.2 ± 4.4	49.3 ± 6.6	2.0 ± 0.0
Pima	74.2 ± 3.9	10.0 ± 3.8	**76.3** ± 5.1	8.1 ± 4.9	74.7 ± 4.0	13.1 ± 10.8
Promoters	83.0 ± 6.4	3.4 ± 1.4	83.0 ± 6.4	3.4 ± 1.4	56.0 ± 6.6	20.0 ± 36.0
Sonar	82.0 ± 6.8	4.6 ± 3.1	78.5 ± 8.9	4.8 ± 2.1	73.5 ± 7.4	5.6 ± 3.6
Soylarge	75.0 ± 5.2	26.3 ± 4.7	75.0 ± 5.2	26.3 ± 4.7	12.3 ± 6.9	2.0 ± 0.0
Soysmall	**97.5** ± 7.5	3.9 ± 0.3	**97.5** ± 7.5	3.9 ± 0.3	30.0 ± 21.8	13.3 ± 18.2
Vehicle	65.1 ± 4.0	24.4 ± 9.6	**65.4** ± 3.5	23.7 ± 5.0	62.1 ± 4.7	52.9 ± 18.6
Votes	95.6 ± 2.6	6.1 ± 2.3	95.6 ± 2.6	6.1 ± 2.3	47.0 ± 8.1	42.5 ± 29.2
Vowel	66.7 ± 7.5	31.2 ± 10.1	65.0 ± 7.7	36.3 ± 8.5	57.9 ± 8.8	39.2 ± 13.5
Wine	95.9 ± 4.6	4.9 ± 0.3	92.4 ± 7.5	4.4 ± 0.7	92.9 ± 5.8	4.4 ± 0.9
Zoo	92.0 ± 7.5	6.2 ± 0.9	92.0 ± 7.5	6.2 ± 0.9	75.0 ± 12.9	33.4 ± 17.4

Table 5.7 Results of Dice, Cosine, Jaccard Coefficient distance metrics.

	Dice coefficient		Cosine coefficient		Jaccard coefficient	
Dataset	*Accuracy*	*Hidden*	*Accuracy*	*Hidden*	*Accuracy*	*Hidden*
2SP	56.8 ± 8.4	4.7 ± 2.7	56.8 ± 7.4	6.4 ± 10.0	55.3 ± 5.9	5.5 ± 3.2
Annealing	-	-	-	-	-	-
Audiology	-	-	-	-	-	-
Bridge	-	-	-	-	-	-
Cancer	-	-	-	-	-	-
CRX	-	-	-	-	-	-
Flag	-	-	-	-	-	-
Glass	66.2 ± 8.9	7.9 ± 4.6	68.6 ± 5.7	11.0 ± 5.4	66.2 ± 8.9	7.9 ± 4.6
Heart	-	-	-	-	-	-
HeartCle	-	-	-	-	-	-
HeartHun	-	-	-	-	-	-
HeartLB	-	-	-	-	-	-
HeartSwi	-	-	-	-	-	-
Hepatitis	-	-	-	-	-	-
Horse	-	-	-	-	-	-
Ionosphere	92.6 ± 3.9	5.4 ± 1.4	**94.3** ± 5.0	5.5 ± 1.6	92.9 ± 3.7	5.8 ± 1.5
Iris	95.3 ± 6.7	3.1 ± 0.5	**97.3** ± 3.3	4.0 ± 0.0	95.3 ± 6.7	3.1 ± 0.5
Liver	66.8 ± 5.8	6.5 ± 5.8	70.6 ± 6.2	6.9 ± 3.7	65.9 ± 5.3	6.2 ± 5.5
Monks-1	-	-	-	-	-	-
Monks-2	-	-	-	-	-	-
Monks-3	-	-	-	-	-	-
Pima	71.6 ± 2.8	13.0 ± 10.2	68.2 ± 5.9	8.0 ± 8.1	72.4 ± 3.0	11.4 ± 8.7
Promoters	-	-	-	-	-	-
Sonar	79.5 ± 7.2	8.2 ± 3.0	76.5 ± 8.1	4.8 ± 2.1	79.0 ± 7.7	6.9 ± 2.3
Soylarge	-	-	-	-	-	-
Soysmall	-	-	-	-	-	-
Vehicle	58.8 ± 3.9	20.2 ± 20.7	61.0 ± 3.3	17.3 ± 7.5	58.7 ± 3.9	20.1 ± 20.8
Votes	-	-	-	-	-	-
Vowel	**69.8** ± 6.4	38.0 ± 8.3	57.3 ± 6.1	35.7 ± 12.1	69.6 ± 7.4	38.1 ± 8.6
Wine	94.3 ± 3.4	6.0 ± 3.6	83.5 ± 6.3	6.7 ± 4.0	80.6 ± 6.5	6.3 ± 4.1
Zoo	-	-	-	-	-	-

Table 5.8 Results of Camberra and Attributed-based distance metrics.

	Camberra		Attribute-based	
Dataset	*Accuracy*	*Hidden*	*Accuracy*	*Hidden*
2SP	*	*	63.7 ± 9.0	16.4 ± 9.2
Annealing	-	-	-	-
Audiology	-	-	-	-
Bridge	-	-	-	-
Cancer	-	-	-	-
CRX	-	-	-	-
Flag	-	-	-	-
Glass	*	*	65.7 ± 8.7	22.7 ± 9.0
Heart	-	-	-	-
HeartCle	-	-	-	-
HeartHun	-	-	-	-
HeartLB	-	-	-	-
HeartSwi	-	-	-	-
Hepatitis	-	-	-	-
Horse	-	-	-	-
Ionosphere	*	*	92.6 ± 4.3	8.5 ± 2.9
Iris	95.3 ± 6.0	3.1 ± 0.3	92.6 ± 4.3	8.5 ± 2.9
Liver	*	*	$\mathbf{72.9} \pm 5.1$	21.5 ± 27.3
Monks-1	-	-	-	-
Monks-2	-	-	-	-
Monks-3	-	-	-	-
Pima	*	*	74.7 ± 3.9	39.5 ± 27.9
Promoters	-	-	-	-
Sonar	77.5 ± 11.2	4.5 ± 2.3	78.5 ± 6.3	5.3 ± 3.4
Soylarge	-	-	-	-
Soysmall	-	-	-	-
Vehicle	*	*	56.9 ± 5.2	76.1 ± 23.7
Votes	-	-	-	-
Vowel	*	*	50.2 ± 6.2	48.7 ± 28.1
Wine	95.3 ± 5.8	4.2 ± 0.8	$\mathbf{97.1} \pm 4.0$	5.5 ± 1.7
Zoo	-	-	-	-

o (overlap) as described in Section 5.2.1. The entries in the tables correspond to means and standard deviations and are shown in the form *mean* $\pm$ *standard deviation*. An '*' indicates that the distance computation was not possible (e.g., the denominator might be zero in Camberra metric) and a '-' indicates that the distance metric was not applicable (e.g., Dice coefficient metric can not be used for nominal or missing values). The best generalization accuracy among different distance metrics are shown in bold face. As we can see from Table 5.3-5.8, no single distance metric outperformed other metrics on all datasets. That is because the performance depends on the distribution of the data. A particular distance metric might be appropriate for certain kinds of datasets while it might not for others. The Euclidean and Manhattan distance metrics outperformed other metrics in many datasets, and gave comparable results to the best ones in other datasets considered.

It is impossible to do a thorough and fair comparison between various learning algorithms since each algorithm has its own *optimal* parameter settings which is usually unknown and not feasible to obtain within a reasonable amount of time. Also, the training and test sets that had been generated and used are not identical in general under the assumption that the experiments have been done a finite number of times. (An infinite number of experiments with random partitions of training and test sets from the same distributions of data can increase the confidence level). Following comparisons (summarized in Table 5.9) should be interpreted in light of those considerations. The best results of DistAI are compared with the best results produced by various learning algorithms in the literature [Weiss & Kapouleas, 1989; Yang & Honavar, 1991; Kohavi, 1994; Lowe, 1995; Andersen & Martinez, 1996; Richeldi & Lanzi, 1996; Parekh, 1998] (Reported in Table 5.9). In particular, the results in [Wilson & Martinez, 1997] are compared separately since they are recent and also obtained by a nearest-neighbor algorithm with a 10-fold cross-validation (NN in Table 5.9). A '-' indicate that the result is not reported in the corresponding reference.

As we can see from Table 5.9, DistAI gave comparable results on most datasets (except **Audiology, Soylarge** and **Vehicle**).

Table 5.9 Comparison of generalization accuracy.

Dataset	DistAl	NN	Reported
2SP	83.7	-	-
Annealing	96.6	96.1	95.6
Audiology	66.0	77.5	77.7
Bridge	63.0	60.6	56.0
Cancer	97.8	95.6	95.9
CRX	87.7	81.5	85.0
Flag	65.8	58.8	-
Glass	70.5	72.4	66.3
Heart	86.7	83.0	74.8
HeartCle	85.3	80.2	77.0
HeartHun	85.9	81.3	77.0
HeartLB	80.0	71.5	79.0
HeartSwi	94.2	93.5	81.0
Hepatitis	84.7	82.6	83.0
Horse	86.0	76.8	80.9
Ionosphere	94.3	92.6	96.7
Iris	97.3	96.0	98.0
Liver	72.9	63.5	69.8
Monks-1	90.9	77.1	100
Monks-2	100	97.5	100
Monks-3	99.1	100	100
Pima	76.3	71.9	76.0
Promoters	88.0	92.4	96.2
Sonar	83.0	87.0	84.7
Soylarge	81.0	92.2	97.1
Soysmall	97.5	100	100
Vehicle	65.4	70.9	79.1
Votes	96.1	95.2	95.2
Vowel	69.8	99.2	61.0
Wine	97.1	97.8	100
Zoo	96.0	98.9	-

The network size of three algorithms (*perceptron cascade* [Burgess, 1994], *cascade correlation* [Fahlman & Lebiere, 1990], *upstart* [Frean, 1990]) for the two spirals problem is shown in [Burgess, 1994]: 17.8 (*perceptron cascade*), 15.2 (*cascade correlation*), 91.4 (*upstart*). DistAl generated more compact networks with 7.7 hidden neurons.

5.4 Summary and Discussion

A fast inter-pattern distance-based constructive learning algorithm, DistAl, is introduced and its performance on a number of datasets is demonstrated. DistAl is different from other constructive learning algorithms in two aspects. First, it does not require an iterative perceptron style weight update rules for determining the connections between neurons. Instead, it computes the distance (using one of the pre-defined distance metrics) between each pattern pair and uses it to set the weights (and the thresholds) between hidden neurons and inputs. The weights between the hidden and output neurons are set using a one-shot (as opposed to iterative) learning algorithm. Thus, DistAl is relatively fast compared in comparison with most neural network training algorithms that rely on an iterative update of weights and consequently require multiple passes through the training set. Furthermore, DistAl is guaranteed to converge to 100% classification accuracy on any non-contradictory training set for most of the distance metrics used in this paper. Second, it generates a single hidden layer composed of *hyperspherical* threshold neurons instead of threshold logic units. Thus, the induced network can potentially discover natural clusters that exist in the data.

Despite its simplicity, experiments reported in this paper show that DistAl yields good performance on almost all real-world datasets that were considered. It also produced good performance on difficult artificial tasks such as parity and the two spirals data which have been used by numerous researchers for evaluation of supervised learning algorithms. In particular, DistAl is suitable to problems that have well-formed clusters and/or certain regularity (e.g., parity) in the pattern space.

A potential disadvantage of DistAl is its need for maintaining the inter-pattern distance matrix during learning. The memory needed to store this matrix grows quadratically with the

size of the training set. This problem can be mitigated by freeing the memory for those patterns that are excluded by a new hidden neuron as learning progresses. It would be interesting to explore variants of DistAI that can avoid the need for maintaining the entire inter-pattern distance matrix during learning.

Because of its speed, DistAI is particularly well-suited to many real-world applications involving large amount of data and/or requesting real-time response such as largescale data mining and knowledge acquisition tasks and hybrid learning systems that use neural network learning as the inner loop of a more complex knowledge discovery process For instance, DistAI, because of its reliance on inter-pattern distances, is sensitive to the presence of irrelevant or misleading attributes in the pattern representation. Consequently, its classification accuracy can be further improved by incorporating a suitable feature subset selection algorithm. This is borne out by the experiments using DistAI in conjunction with a genetic algorithm for feature subset selection [Yang & Honavar, 1998a; Yang & Honavar, 1998b]. See Chapter 6 for details.

DistAI is also a good candidate for knowledge-based theory refinement. The domain knowledge can be translated into the initial network and then refined using DistAI. See Chapter 7 for details.

Some avenues for future research include:

- Each constructive learning algorithm has its own set of inductive and representational biases implicit in the design choices that determine when and where a new neuron is added and how it is trained. A systematic characterization of this bias would be useful in guiding the design of constructive algorithms exhibiting improved performance.

- A systematic comparison of the performance of the constructive learning algorithms against the performance of other classification algorithms (e.g., algorithms in chapter 4), backpropagation learning algorithm [Rumelhart *et al.*, 1986], decision tree algorithms [Quinlan, 1986; Quinlan, 1993], nearest neighbor algorithms [Duda & Hart, 1973; Dasarathy, 1991], etc.) would be useful in gaining a better understanding of their relative advantages and disadvantages.

- The constructive learning algorithms can be modified for *incremental learning.* This is significant for learning wherein a single network is trained over a period of time to perform different tasks so that each task can exploit the useful regularities about the environment discovered by the network in the course of learning to perform the previous tasks. Also, constructive learning algorithms can offer framework for knowledge-based theory refinement. The domain theory can be incorporated into the initial network architecture and then refined by a constructive learning algorithm. (See chapter 7 for some approaches to this purpose).

- All the constructive neural network learning algorithms considered in this dissertation are for pattern classification tasks. Adaptation of these algorithms for function approximation is also of interest.

- A cross-validation based criterion for training constructive networks must be employed wherein the training is stopped when the network's generalization begins to deteriorate after the addition of a new neuron (or a group of neurons). It is likely to generate compact networks that exhibit good generalization properties with relatively little training as opposed to the current stopping-criterion of zero classification errors which might lead to over-fitting of the training set. This is verified in our comparative study of the performance of DistAI with other algorithms (Table 5.9).

- The theoretical analysis of DistAI is of significant interest. For instance, the sample complexity of DistAI to obtain a certain degree of classification accuracy can be explored in the PAC-learning framework [Valiant, 1984].

- It is often the case that the generalization performance of inductive learning algorithms can be substantially improved by augmenting them with suitable algorithms for selecting a relevant subset of a much larger set of input attributes many of which might be irrelevant or misleading. Exploration of constructive learning algorithms augmented with suitable feature subset selection techniques will thus be of interest. In chapter 6, we demonstrate the effectiveness of genetic algorithms for feature subset selection for pat-

tern classification using DistAI [Yang & Honavar, 1997; Yang & Honavar, 1998a; Yang & Honavar, 1998b].

6 FEATURE SUBSET SELECTION USING A GENETIC ALGORITHM

6.1 Introduction

Many practical pattern classification tasks (e.g., medical diagnosis) require learning of an appropriate classification function that assigns a given input pattern (typically represented using a vector of attribute or feature values) to one of a finite set of classes. The choice of features, attributes, or measurements used to represent patterns that are presented to a classifier affect (among other things):

- The accuracy of the classification function that can be learned using an inductive learning algorithm (e.g., a decision tree induction algorithm or a neural network learning algorithm): The features used to describe the patterns implicitly define a pattern language. If the language is not expressive enough, it would fail to capture the information that is necessary for classification and hence regardless of the learning algorithm used, the accuracy of the classification function learned would be limited by this lack of information.

- The time needed for learning a sufficiently accurate classification function: For a given representation of the classification function, the features used to describe the patterns implicitly determine the search space that needs to be explored by the learning algorithm. An abundance of irrelevant features can unnecessarily increase the size of the search space, and hence the time needed for learning a sufficiently accurate classification function.

- The number of examples needed for learning a sufficiently accurate classification function: All other things being equal, the larger the number of features used to describe the patterns in a domain of interest, the larger is the number of examples needed to learn a classification function to a desired accuracy [Langley, 1995; Mitchell, 1997].

- **The cost of performing classification using the learned classification function: In many practical applications e.g., medical diagnosis, patterns are described using observable symptoms as well as results of diagnostic tests. Different diagnostic tests might have different costs as well as risks associated with them. For instance, an invasive exploratory surgery can be much more expensive and risky than say, a blood test.**

- The comprehensibility of the knowledge acquired through learning: A primary task of an inductive learning algorithm is to extract *knowledge* (e.g., in the form of classification rules) from the training data. Presence of a large number of features, especially if they are irrelevant or misleading, can make the knowledge difficult to comprehend by humans. Conversely, if the learned rules are based on a small number of relevant features, they would be much more concise and hence easier to understand, and use by humans.

This presents us with a *feature subset selection problem* in automated design of pattern classifiers. The feature subset selection problem refers the task of identifying and selecting a useful subset of features to be used to represent patterns from a larger set of often mutually redundant, possibly irrelevant, features with different associated measurement costs and/or risks. An example of such a scenario which is of significant practical interest is the task of selecting a subset of clinical tests (each with different financial cost, diagnostic value, and associated risk) to be performed as part of a medical diagnosis task. Other examples of feature subset selection problem include large scale data mining applications, power system control [Zhou *et al.*, 1997], construction of user interest profiles for text classification [Yang *et al.*, 1998d] and sensor subset selection in the design of autonomous robots [Balakrishnan & Honavar, 1996a].

The rest of this chapter is organized as follows: Section 6.2 summarizes various approaches to the feature subset selection. Section 6.3 describes our approach that uses a genetic algorithm for neural network pattern classifiers. Section 6.4 explains the implementation details in our experiments. Section 6.5 presents the results of various experiments designed to evaluate the performance of our approach on some benchmark classification problems as well as a document classification task. Section 6.6 concludes with summary and discussion of some directions for

future research.

6.2 Approaches to Feature Subset Selection

A number of approaches to feature subset selection have been proposed in the literature. (See [Siedlecki & Sklansky, 1988; Doak, 1992; Langley, 1994; Dash & Liu, 1997] for surveys). These approaches involve searching for an optimal subset of features based on some criteria of interest. Feature subset selection problem can be viewed as a special case of the *feature weighting* problem. It involves assigning a real-valued weight to each feature. The weight associated with a feature measures its relevance or significance in the classification task [Cost & Salzberg, 1993; Punch *et al.*, 1993; Wettschereck *et al.*, 1995]. If we restrict the weights to be binary valued, the feature weighting problem reduces to the feature subset selection problem. The focus of this paper is on feature subset selection.

Let $\mu(S)$ be a performance measure that is used to evaluate a feature subset S with respect to the criteria of interest (e.g., cost and accuracy of the resulting classifier). Feature subset selection problem is essentially an optimization problem which involves searching the space of possible feature subsets to identify one that is optimal or near-optimal with respect to μ. Feature subset selection algorithms can broadly be classified into three categories according to the characteristics of the search strategy employed.

6.2.1 Feature Subset Selection Using Exhaustive Search

In this approach, the candidate feature subsets are evaluated with respect to the performance measure μ and an *optimal* feature subset is found using exhaustive search. The Focus algorithm [Almuallim & Dietterich, 1994] employs the breadth-first search algorithm to find the minimal combination of features sufficient to construct a hypothesis that is consistent with the training examples. The algorithm proposed by [Sheinvald *et al.*, 1990] uses the *minimum description length* criterion [Rissanen, 1978] to select an optimal feature subset using exhaustive enumeration and evaluation of candidate feature subsets. Exhaustive search is computationally infeasible in practice, except in those rare instances where the total number of features is

quite small.

6.2.2 Feature Subset Selection Using Heuristic Search

Since exhaustive search over all possible subsets of a feature set is not computationally feasible in practice, a number of authors have explored the use of *heuristics* for feature subset selection, often in conjunction with branch and bound search, a technique that is well–known in combinatorial optimization [Cormen *et al.*, 1990] and artificial intelligence [Russell & Norvig, 1995]. *Forward selection* and *backward elimination* are the most common sequential branch and bound search algorithms used in feature subset selection [Narendra & Fukunaga, 1977; Devijver, 1982; Foroutan & Sklansky, 1987; Fukunaga, 1990]. Forward selection starts with an empty feature set and adds a feature at a time, at each stage choosing the addition that most increases μ. Backward elimination starts with the entire feature set and at each step drops the feature whose absence least decreases μ. Both forward and backward selection procedures are optimal at each stage, but are unable to anticipate complex *interactions* between features that might affect the performance of the classifier. A related approach, called the *exchange strategy* starts with an initial feature subset (perhaps found by forward selection or backward elimination) and then tries to exchange a feature in the selected subset with one of the features that is outside it. We can often find a feature subset that is guaranteed to be the best for a given size of the feature subset without considering all possible subsets using branch and bound search [Narendra & Fukunaga, 1977] if we assume that μ is monotone. That is, adding features is guaranteed to not decrease μ. It is worth pointing out that in many practical pattern classification scenarios, the monotonicity assumption is not satisfied. For example, addition of irrelevant features (e.g., social security numbers in medical records in a diagnosis task) can significantly worsen the generalization accuracy of a decision tree classifier [Quinlan, 1993]. Furthermore, feature subset selection techniques that rely on the monotonicity of the performance criterion, although they appear to work reasonably well with linear classifiers, can exhibit poor performance with non-linear classifiers such as neural networks [Ripley, 1996].

The use of systematic search to find a feature subset that is consistent with training data

by forward selection using a reliability measure is reported in [Schlimmer, 1993]. Five greedy hillclimbing procedures (with different sequential search methods) for obtaining good generalization with decision tree construction algorithms (ID3 and C4.5) [Quinlan, 1993] were proposed in [Caruana & Freitag, 1994]. In related work, [John *et al.*, 1994] used both forward selection and backward elimination to minimize the cross validation error of decision tree classifiers [Quinlan, 1993]; [Kohavi, 1994; Kohavi & Frasca, 1994] used hillclimbing and best-first search for feature subset selection for decision tree classifiers. Koller et al. [Koller & Sahami, 1996; Koller & Sahami, 1997] used forward selection and backward elimination to select a feature that is subsumed by the remaining features (determined by the *Markov blanket*, the set of features that render the selected feature conditionally independent of the remaining features) for constructing Naive Bayesian [Duda & Hart, 1973; Mitchell, 1997] and decision tree classifiers [Quinlan, 1993]. The Preset algorithm [Modrzejewski, 1993] employs the *rough set theory* [Pawlak, 1991] to select a feature subset by rank ordering the features to generate a minimal decision tree. A class of techniques based for feature subset selection using the probability of error and correlation among features is reported in [Mucciardi & Gose, 1971].

6.2.3 Feature Subset Selection Using Randomized Search

Randomized algorithms [Motwani & Raghavan, 1996] make use of randomized or probabilistic (as opposed to deterministic) steps or sampling processes. Several researchers have explored the use of such algorithms for feature subset selection. The Relief algorithm [Kira & Rendell, 1992] assigns weights to features (based on their estimated effectiveness for classification) using the randomly sampled instances. Features whose weights exceed a user-determined threshold are selected in designing the classifier. Several extensions of Relief have been introduced to handle noisy or missing features as well as multi-category classification [Kononenko, 1994]. A randomized hillclimbing search for feature subset selection for nearest neighbor classifiers [Cover & Hart, 1967; Diday, 1974; Dasarathy, 1991] was proposed in [Skalak, 1994]. The LVF and LVW algorithms [Liu & Setiono, 1996b; Liu & Setiono, 1996a] are randomized algorithms that generate several random feature subsets and pick the one that has the least

number of *unfaithful* patterns in the space defined by the feature subset (LVF) or the one that has the lowest error using a decision tree classifier (LVW) giving preference to smaller feature subsets. (Two patterns are said to be *unfaithful* if they have the same feature values but different class labels). Several authors have explored the use of randomized population-based heuristic search techniques such as genetic algorithms (GA) for feature subset selection for decision tree and nearest neighbor classifiers [Siedlecki & Sklansky, 1989; Brill *et al.*, 1992; Punch *et al.*, 1993; Richeldi & Lanzi, 1996] or rule induction systems [Vafaie & De Jong, 1993]. A related approach used *lateral feedback* networks [Guo, 1992; Kothari & Agyepong, 1996] to evaluate feature subsets [Guo & Uhrig, 1992]. Feature subset selection techniques that employ genetic algorithms do not require the restrictive monotonicity assumption. They also readily lend themselves to the use of multiple selection criteria (e.g., classification accuracy, feature measurement cost, etc.). This makes them particularly attractive in the design of pattern classifiers in many practical scenarios.

6.2.4 Filter and Wrapper Approaches to Feature Subset Selection

Feature subset selection algorithms can also be classified into two categories based on whether or not feature selection is done independently of the learning algorithm used to construct the classifier. If feature selection is performed independently of the learning algorithm, the technique is said to follow a *filter* approach. Otherwise, it is said to follow a *wrapper* approach [John *et al.*, 1994]. While the filter approach is generally computationally more efficient than the wrapper approach, its major drawback is that an optimal selection of features may not be independent of the inductive and representational biases of the learning algorithm that is used to construct the classifier. The wrapper approach on the other hand, involves the computational overhead of evaluating candidate feature subsets by executing a selected learning algorithm on the dataset represented using each feature subset under consideration. This is feasible only if the learning algorithm used to train the classifier is relatively fast. Figure 6.1 summarizes the filter and wrapper approaches. The approach to feature subset selection proposed in this paper is an instance of the wrapper approach. It utilizes a genetic algorithm

for feature subset selection. Feature subsets are evaluated by computing the generalization accuracy of (and optionally cost of features used in) the neural network classifier constructed using a computationally efficient neural network learning algorithm called DistAl [Yang *et al.*, 1998b].

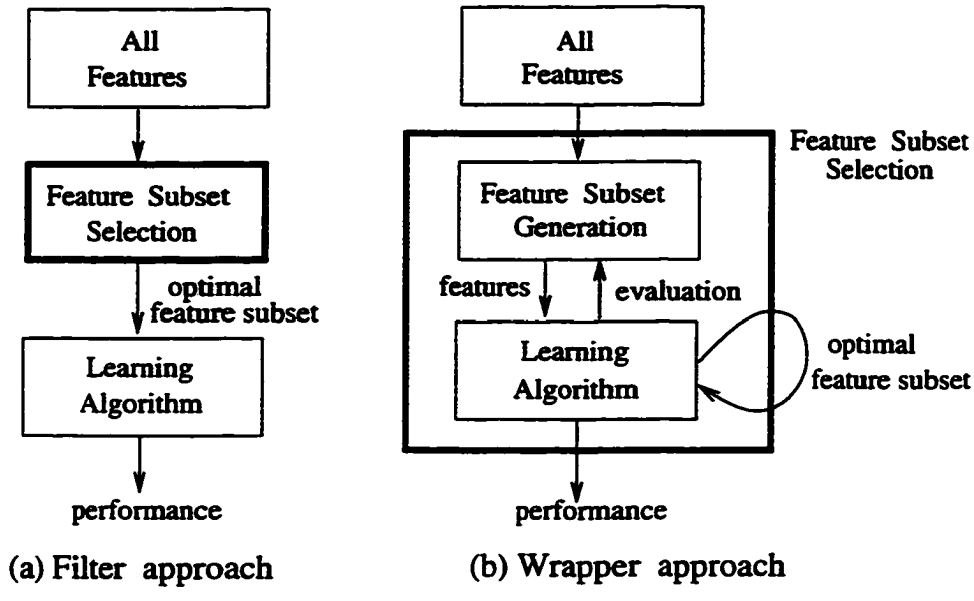


Figure 6.1 *Filter* and *Wrapper* approaches to feature subset selection.

6.3 Feature Selection Using a Genetic Algorithm for Neural Network Pattern Classifiers

Feature subset selection in the context of many practical problems (e.g., diagnosis) presents an instance of a multi-criteria optimization problem. The multiple criteria to be optimized include the accuracy of classification, cost and risk associated with classification which in turn depends on the selection of features used to describe the patterns. Genetic algorithms offer a particularly attractive approach for multi-criteria optimization.

Neural networks offer an attractive framework for the design of trainable pattern classifiers for real-world real-time pattern classification tasks on account of their potential for parallelism and fault and noise tolerance, [Gallant, 1993; Honavar, 1994; Hassoun, 1995; Ripley, 1996;

Mitchell, 1997; Honavar *et al.*, 1999a; Honavar *et al.*, 1999b]. (See chapter 2 for detailed description on artificial neural networks).

While genetic algorithms are generally quite effective for rapid global search of large search spaces in difficult optimization problems, neural networks offer a particularly attractive approach to finetuning promising solutions once they have been identified. Thus, it is attractive to explore combinations of global and local search techniques in the solution of difficult design or optimization problems [Mitchell, 1996]. Against this background, the use of genetic algorithms for feature subset selection in the design of neural network pattern classifiers is clearly of interest.

This chapter explores GADistAI, a wrapper-based multi-criteria approach to feature subset selection using a genetic algorithm in conjunction with DistAI. However, the general approach can be used with any inductive learning algorithm. The interested reader is referred to [Honavar, 1994; Langley, 1995; Mitchell, 1997; Honavar *et al.*, 1999a; Honavar *et al.*, 1999b] for surveys of different approaches to inductive learning.

6.3.1 Genetic Algorithms

Evolutionary algorithms [Goldberg, 1989; Holland, 1992; Koza, 1992; Fogel, 1995; Michalewicz, 1996; Mitchell, 1996; Banzaf *et al.*, 1997] include a class related randomized, population-based heuristic search techniques which include genetic algorithms [Goldberg, 1989; Holland, 1992; Mitchell, 1996], genetic programming [Koza, 1992; Banzaf *et al.*, 1997], evolutionary programming [Fogel, 1995], and variety of related approaches [Michalewicz, 1996; Mitchell, 1996]. They are inspired by processes that are modeled after biological evolution. Central to such evolutionary systems is the idea of a population of potential solutions (individuals) that corresponds to members of a high-dimensional search space.

The individuals represent candidate solutions to the optimization problem being solved. A wide range of genetic representations (e.g., bit vectors, LISP programs, matrices, etc.) can be used to encode the individuals depending on the space of solutions that needs to be searched. In genetic algorithms [Goldberg, 1989; Michalewicz, 1996; Mitchell, 1996], the individuals are

typically represented by n-bit binary vectors. The resulting search space corresponds to an n-dimensional boolean space. In the feature subset selection problem, each individual would represent a feature subset.

It is assumed that the quality of each candidate solution (or fitness of the individual in the population) can be evaluated using a fitness function. In the feature subset selection problem, the fitness function would evaluate the selected features with respect to some criteria of interest (e.g., cost of the resulting classifier, classification accuracy of the classifier, etc.). In this case, it is essentially the μ function defined earlier.

Evolutionary algorithms use some form of fitness-dependent probabilistic selection of individuals from the current population to produce individuals for the next generation. A variety of selection techniques have been explored in the literature. Some of the most common ones are *fitness-proportionate* selection, *rank-based* selection, and *tournament-based* selection [Goldberg, 1989; Michalewicz, 1996; Mitchell, 1996]. The selected individuals are subjected to the action of genetic operators to obtain new individuals that constitute the next generation. The genetic operators are usually designed to exploit the known properties of the genetic representation, the search space, and the optimization problem to be solved. Genetic operators enable the algorithm to *explore* the space of candidate solutions. See [Balakrishnan & Honavar, 1995] for a discussion of some desirable properties of genetic representations and operators.

Mutation and *crossover* are two of the most commonly used operators that are used with genetic algorithms that represent individuals as binary strings. Mutation operates on a single string and generally changes a bit at random. Thus, a string 11010 may, as a consequence of random mutation, get changed to 11110. Crossover, on the other hand, operates on two parent strings to produce two offspring. With a randomly chosen crossover position 4, the two strings 01101 and 11000 yield the offspring 01100 and 11001 as a result of crossover. Other genetic representations (e.g., matrices, LISP programs) require the use of appropriately designed genetic operators [Michalewicz, 1996; Mitchell, 1996; Banzaf *et al.*, 1997].

The process of fitness-dependent selection and application of genetic operators to generate successive generations of individuals is repeated many times until a satisfactory solution

is found (or the search fails). It can be shown that evolutionary algorithms of the sort outlined above simulate highly opportunistic and exploitative randomized search that explores high-dimensional search spaces rather effectively under certain conditions [Holland, 1992]. In practice, the performance of evolutionary algorithms depends on a number of factors including: the choice of genetic representation and operators, the fitness function, the details of the fitness-dependent selection procedure, and the various user-determined parameters such as population size, probability of application of different genetic operators, etc. The specific choices made in the experiments reported in this paper are summarized in section 6.4.

6.3.2 Genetic Algorithm Wrapper approach to Feature Subset Selection for Neural Network Pattern Classifiers: Some Practical Considerations

Genetic algorithms offer an attractive technique for feature subset selection for neural network pattern classifiers for several reasons, some of which were mentioned above. However, we are faced with several difficulties in using this approach in practice.

Traditional neural network learning algorithms (e.g., backpropagation) perform an error gradient guided search for a suitable setting of weights in the weight space determined by a user-specified network architecture. This ad hoc choice of network architecture often inappropriately constrains the search for an appropriate setting of weights. For example, if the network has fewer neurons than necessary, the learning algorithm will fail to find the desired classification function. If the network has far more neurons than necessary, it can result in overfitting of the training data leading to poor generalization. In either case, it would make it difficult to evaluate the usefulness of a feature subset employed to describe (or represent) the training patterns used to train the neural network.

Gradient based learning algorithms although mathematically well-founded for unimodal search spaces, can get caught in local minima of the error function. This can complicate the evaluation of a feature subset employed to represent the training patterns used to train the neural networks. This is due to the fact that the poor performance of the classifier might be due to the failure of the learning algorithm, and not the feature subset used.

Fortunately, constructive neural network learning algorithms [Gallant, 1993; Honavar & Uhr, 1993; Honavar *et al.*, 1999a] eliminate the need for ad hoc, and often inappropriate a-priori choices of network architectures; and can potentially discover near-minimal networks whose size is commensurate with the complexity of the classification task that is implicitly specified by the training data. Several new, provably convergent, and relatively efficient constructive learning algorithms for multi-category real as well as discrete valued pattern classification tasks have begun to appear in the literature [Yang *et al.*, 1996; Parekh *et al.*, 1997a; Parekh *et al.*, 1997b; Yang *et al.*, 1998b; Honavar *et al.*, 1999a]. Many of these algorithms have demonstrated very good performance in terms of reduced network size, learning time, and generalization in a number of experiments with both artificial and fairly large real-world datasets. [Honavar & Uhr, 1993; Parekh *et al.*, 1997a; Yang *et al.*, 1998b]. However, most of them, with the exception of DistAI [Yang *et al.*, 1998b] (in Chapter 5) use time–consuming iterative training algorithms for setting the weights of the neurons.

Using genetic algorithms for feature subset selection for the design of neural network pattern classifiers involves running a genetic algorithm for several generations. In each generation, evaluation of an individual (a feature subset) requires training the corresponding neural network and computing its accuracy and cost. This evaluation has to be performed for each of the individuals in the population. Thus, it is not feasible to use computationally expensive iterative weight update algorithms for training neural network classifiers for evaluating candidate feature subsets. Against this background, DistAI offers an attractive approach to training neural networks. Key steps in our approach are shown in Figure 6.2: Starting from the initial population (of candidates having different feature subsets), new populations are generated repeatedly from the previous ones by applying genetic operators (i.e., crossover and mutation) to the selected parents, evaluating the fitness values of offsprings by DistAI and ranking them according to their fitness values. The best individual is obtained after the last generation.

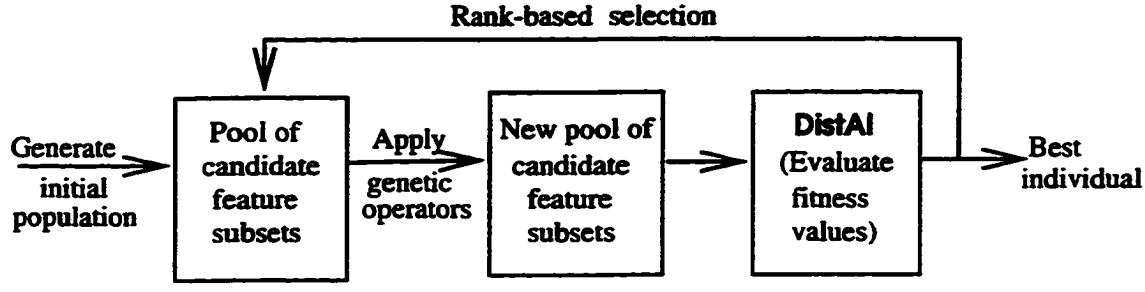


Figure 6.2 GADistAI: feature subset selection using a genetic algorithm with DistAI.

6.4 Implementation Details

As explained earlier, the use of a genetic algorithm in any search or optimization problem requires:

- the choice of a representation for encoding candidate solutions to be manipulated by the genetic algorithm
- the definition of a fitness function that is used to evaluate the candidate solutions
- the definition of a selection-scheme (e.g., fitness-proportionate selection)
- the definition of suitable genetic operators that are used to transform candidate solutions (and thereby explore the search space)
- setting of user-controlled parameters (e.g., probability of applying a particular genetic operator, size of the population, etc.)

Our experiments were run using a genetic algorithm [Goldberg, 1989; Mitchell, 1996] using rank-based selection strategy. The probability of selection of the highest ranked individual is p (where $0.5 < p < 1.0$ is a user-specified parameter), that of the second highest ranked individual is $p(1-p)$, that of the third highest ranked individual is $p(1-p)^2$,..., that of the last ranked individual is 1−(sum of the probabilities of selection of all the other individuals). The rank-based selection strategy gives a non-zero probability of selection of each individual [Mitchell, 1996]. Our experiments used the following parameter settings:

- Population size: 50
- Number of generation: 20
- Probability of crossover: 0.6
- Probability of mutation: 0.001
- Probability of selection of the highest ranked individual: 0.6

The parameter settings were based on results of several preliminary runs. They are comparable to the typical values mentioned in the literature [Mitchell, 1996].

Each individual in the population represents a candidate solution to the feature subset selection problem. Let m be the total number of features available to choose from to represent the patterns to be classified. In a medical diagnosis task, these would be observable symptoms and a set of possible diagnostic tests that can be performed on the patient. It is represented by a binary vector of dimension m (where m is the total number of features). If a bit is a 1, it means that the corresponding feature is selected. A value of 0 indicates that the corresponding feature is not selected. The fitness of an individual is determined by evaluating the neural network constructed by DistAI using a training set whose patterns are represented using only the selected subset of features. If an individual has n bits turned on, the corresponding neural network has n input nodes.

The fitness function has to combine two different criteria – the accuracy of the classification function realized by the neural network and the cost of performing classification. The accuracy of the classification function can be estimated by calculating the percentage of patterns in a test set that are correctly classified by the neural network in question. A number of different measures of the cost of classification suggest themselves: cost of measuring the value of a particular feature needed for classification (or the cost of performing the necessary test in a medical diagnosis application), the risk involved, etc. To keep things simple, we chose a 2-criteria fitness function defined as follows:

$$fitness(x) = accuracy(x) - \frac{cost(x)}{accuracy(x) + 1} + cost_{max} \tag{6.1}$$

where $fitness(x)$ is the fitness of the feature subset represented by x, $accuracy(x)$ is the test accuracy of the neural network classifier trained using DistAl using the feature subset represented by x, $cost(x)$ is the sum of measurement costs of feature subset represented by x, and $cost_{max}$ is an upper bound on the costs of candidate solutions. In this case, it is simply the sum of the costs associated with all of the features. This is clearly a somewhat ad hoc choice. However, it does discourage trivial solutions (e.g., a zero cost solution with a very low accuracy) from being selected over reasonable solutions which yield high accuracy at a moderate cost. It also ensures that $\forall x\ 0 \leq fitness(x) \leq (100 + cost_{max})$. In practice, defining suitable tradeoffs between the multiple objectives has to be based on knowledge of the domain. In general, it is a non-trivial task to combine multiple optimization criteria into a single fitness function. A wide variety of approaches have been examined in the utility theory literature [Keeney & Raiffa, 1976].

6.5 Experiments

6.5.1 Datasets

The experiments reported here used a wide range of real-world datasets from the machine learning data repository at the University of California at Irvine [Murphy & Aha, 1994] as well as a carefully constructed artificial dataset (3-bit parity) to explore the feasibility of using genetic algorithms for feature subset selection for neural network classifiers. The feature subset selection using DistAl is also applied to document classification problem for journal paper abstracts and news articles.

- *3-bit Parity Dataset*

  This dataset was constructed to explore the effectiveness of the genetic algorithm in selecting an appropriate subset of relevant features in the presence of redundant features so as to minimize the cost and maximize the accuracy of the resulting neural network pattern classifier. The modified training set is constructed as follows: The original features are replicated once (to introduce redundancy) thereby doubling the number

of features. Then an additional set of irrelevant features are generated and are assigned random boolean values. 100 7-bit random vectors were generated and augmented with the 6-bit vectors (corresponding to the original 3 bits plus an identical set of 3 bits). Each feature in the resulting dataset is assigned a random cost between 0 and 9. The performance considering the random costs in addition to the accuracy (see equation (6.1)) was compared with that obtained by considering the accuracy alone. Table 6.1 includes this dataset. Size is the number of patterns in the dataset, Dimension is the number of input attributes, Missing? is whether there are any missing values, and Class is the number of output classes in Table 6.1.

Table 6.1 Parity and document datasets used in the experiments.

Dataset	Size	Dimension	Attribute Type	Missing?	Class
3-bit parity problem (**3P**)	100	13	numeric	No	2
paper abstracts 1 (**Abstract1**)	100	790	numeric	No	2
paper abstracts 2 (**Abstract2**)	100	790	numeric	No	2
news articles 1 (**Reuters1**)	939	1568	numeric	No	6
news articles 2 (**Reuters2**)	139	435	numeric	No	4
news articles 3 (**Reuters3**)	834	1440	numeric	No	8

- *Datasets from UCI Repository*

  In our experiments with real world datasets, our objective was to compare the neural networks built using feature subsets selected by the genetic algorithm with those that use the entire set of features available. Some of the datasets (with reasonable size of feature sets) were chosen in our experiment from Table 5.1. Some medical datasets include measurement costs for the features, but most of the datasets lack this information. Therefore, our experiments with the datasets from UCI repository focused on identifying a minimal subset of features that yield high accuracy neural network classifiers. Where measurement costs were available, the performance considering the cost in addition to the accuracy was compared with that obtained by considering the accuracy alone.

- *Document Datasets*

  The paper abstracts were chosen from three different sources: IEEE Expert magazine, Journal of Artificial Intelligence Research and Neural Computation. The news articles were obtained from Reuters dataset. Each document is represented in the form of a vector of numeric weights for each of the words (terms) in the vocabulary. The weights correspond to the term frequency and inverse document frequency (TFIDF) [Salton & McGill, 1983; Yang *et al.*, 1998d] values for the corresponding words (see also section 8.3.2). The training sets for paper abstracts were generated based on the classification of the corresponding documents into two classes (interesting and not interesting) by two different individuals, resulting in two different data sets (**Abstract1** and **Abstract2**). The classifications for news articles were given based on their topics (6, 4 and 8 classes) following [Koller & Sahami, 1997], resulting in three different datasets (**Reuters1**, **Reuters2** and **Reuters3**), respectively. These datasets are also summarized in Table 6.1. Since these datasets do not have measurement costs for the features, our experiments with document datasets also focused on identifying a minimal subset of features that yield high accuracy neural network classifiers.

6.5.2 Experimental Results

Two different sets of experiments were run to explore the performance of GADistAl. The first set of experiments were designed to explore the effect of feature subset selection on the performance of DistAl on a given choice of training and test sets. Each dataset was randomly partitioned into a training and test set (with 90% of the data used for training and the remaining 10% for testing). The genetic algorithm was used to select the best feature subset on the basis of this choice of training and test sets. The results were averaged over 5 independent runs of the genetic algorithm, for a given choice of training and test set. This process was repeated 10 times with 10 different choices of training and test set, which makes $5 \times 10 = 50$ runs of the genetic algorithm.

The second set of experiments explored a somewhat different, but related question. Since

feature subset selection in GADistAI is guided by the fitness function, it seems reasonable to expect that the quality of fitness estimates will have some impact on the performance of DistAI. Thus, it is interesting to explore the performance of GADistAI when the fitness estimates are obtained using several training and test sets. Thus, in this set of experiments, fitness estimates used by GADistAI were obtained by averaging the observed fitness values for 10 different partitions of the data into training and test sets. The reported results represent averages over 5 independent runs of the algorithm.

The results of those two different sets of experiments are shown in Tables 6.2-6.6. The entries in the tables give the means (and standard deviations) in the form *mean* ($\pm$ *standard deviation*).

6.5.2.1 Improvement in Generalization using Feature Subset Selection

To study the effect of feature subset selection on generalization, experiments were run using classification accuracy as the fitness function. The results in Table 6.2 indicate that the networks constructed using GA-selected subset of features compare quite favorably with networks that use all of the features in all randomly partitioned datasets. In particular, feature subset selection resulted in substantial improvement in generalization on many of the datasets. (For example, 100% accuracy were yielded in **P3, Promoters**, and **Zoo** datasets). Also, the number of features selected is significantly smaller than the total number of features present in the original data representation in all of the datasets.

The results shown in Table 6.3 indicate that the networks constructed using GA-selected subset of features are comparable to the networks that use all of the features in most of the datasets with 10-fold cross-validation. Clearly, GADistAI outperformed plain DistAI (with all features) in the parity problem in the sense that it successfully selected important features giving 100% generalization. For the remaining datasets, the improvement is generalization ranged from modest in some cases to marginal in others. The best individual generated by GADistAI outperformed DistAI in almost all datasets. Again, the number of features selected is significantly smaller than the total number of features present in the original data representation

Table 6.2 Results for randomly partitioned datasets.

	DistAl		GADistAl	
Dataset	*Features*	*Accuracy*	*Features*	*Accuracy*
3P	13	79.0±12.2	6.6 ± 1.6	100 ± 0.0
Annealing	38	96.6±2.0	21.0 ± 3.1	99.5 ± 0.9
Audiology	69	66.0±9.7	36.4 ± 3.5	83.5 ± 8.2
Bridges	11	63.0 ± 7.8	5.6 ± 1.5	81.6 ± 7.6
Cancer	9	97.8 ± 1.2	5.4 ± 1.4	99.3 ± 0.9
CRX	15	87.7 ± 3.3	8.0 ± 2.1	91.5 ± 2.8
Flag	28	65.8 ± 9.5	14.0 ±2.6	78.1 ±7.8
Glass	9	70.5 ± 8.5	5.5 ±1.4	80.8 ±5.0
Heart	13	86.7 ± 7.6	7.2 ±1.6	93.9 ±3.8
HeartCle	13	85.3 ± 2.7	7.3 ±1.7	92.9 ±3.6
HeartHun	13	85.9 ± 6.3	7.0 ±1.2	93.0 ±4.0
HeartLB	13	80.0 ± 7.4	7.1 ±1.7	91.0 ±5.7
HeartSwi	13	94.2 ± 3.8	6.6 ±1.7	98.3 ±3.3
Hepatitis	19	84.7 ± 9.5	9.2 ±2.3	97.1 ±4.3
Horse	22	86.0 ± 3.6	11.1 ±2.3	92.6 ±3.4
Ionosphere	34	94.3 ± 5.0	17.3 ±3.5	98.6 ±2.4
Pima	8	76.3 ± 5.1	3.8 ±1.5	79.5 ±3.1
Promoters	57	88.0 ± 7.5	28.8 ±3.3	100 ±0.0
Sonar	60	83.0 ± 7.8	30.7 ±3.7	97.2 ±2.9
Soybean	35	81.0 ± 5.6	19.4 ± 2.7	92.8 ±5.9
Vehicle	18	65.4 ± 3.5	9.1 ±1.7	68.8 ±4.3
Votes	16	96.1 ± 1.5	8.9 ±1.8	98.8 ±1.2
Vowel	10	69.8 ± 6.4	6.5 ±1.2	78.4 ±3.8
Wine	13	97.1 ± 4.0	6.7 ±1.6	99.4 ±2.1
Zoo	16	96.0 ± 4.9	9.3 ±1.6	100 ±0.0
Abstract1	790	89.0±9.4	393.7 ± 12.9	97.6 ± 4.7
Abstract2	790	84.0±12.0	393.8 ± 14.6	94.4 ± 7.3
Reuters1	1568	91.6±2.9	786.1 ± 19.1	94.9 ± 2.5
Reuters2	435	88.5±10.5	218.3 ± 9.7	97.5 ± 4.7
Reuters3	1440	96.4±1.6	715.4 ± 20.3	98.7 ± 1.0

Table 6.3 Results for 10-fold cross-validation.

	DistAI		GADistAI (average)		GADistAI (best)	
Dataset	*Features*	*Accuracy*	*Features*	*Accuracy*	*Features*	*Accuracy*
3P	13	79.0±12.2	4.8 ± 0.7	100 ± 0.0	4	100 ± 0.0
Annealing	38	96.6±2.0	20.0 ± 1.4	98.8 ± 0.4	18	99.5 ± 1.2
Audiology	69	66.0±9.7	37.2 ± 1.8	72.6 ± 2.8	39	76.5 ± 13.8
Bridges	11	63.0 ± 7.8	4.9 ± 0.6	56.9 ± 7.6	5	67.0 ± 11.9
Cancer	9	97.8 ± 1.2	6.0 ± 1.1	98.0 ± 0.3	8	98.6 ± 0.9
CRX	15	87.7 ± 3.3	7.4 ± 2.6	87.7 ± 0.4	6	88.0 ± 2.8
Flag	28	65.8 ± 9.5	14.2 ± 2.8	63.9 ±6.1	18	70.0 ± 8.8
Glass	9	70.5 ± 8.5	4.4 ± 0.8	69.3 ±2.5	5	71.0 ± 9.4
Heart	13	86.7 ± 7.6	7.6 ± 0.8	85.5 ±0.7	7	85.9 ± 5.4
HeartCle	13	85.3 ± 2.7	8.4 ± 0.8	86.9 ±0.6	9	87.7 ± 4.0
HeartHun	13	85.9 ± 6.3	7.4 ± 1.4	85.4 ±1.3	8	87.2 ± 2.2
HeartLB	13	80.0 ± 7.4	7.6 ± 1.0	79.8 ±1.9	6	83.0 ± 6.0
HeartSwi	13	94.2 ± 3.8	7.4 ± 1.7	95.3 ±1.1	8	96.7 ± 4.1
Hepatitis	19	84.7 ± 9.5	10.2 ± 1.6	85.2 ±2.9	10	88.7 ± 9.5
Horse	22	86.0 ± 3.6	9.6 ± 2.7	83.2 ±1.6	5	85.0 ± 7.0
Ionosphere	34	94.3 ± 5.0	16.6 ± 3.0	94.5 ± 0.8	13	96.0 ± 4.3
Pima	8	76.3 ± 5.1	4.0 ± 1.7	73.1 ±3.1	2	76.8 ± 3.8
Promoters	57	88.0 ± 7.5	30.6 ± 2.1	89.8 ±1.7	31	92.0 ± 7.5
Sonar	60	83.0 ± 7.8	32.2 ± 2.2	84.0 ±1.6	28	85.5 ± 7.6
Soybean	35	81.0 ± 5.6	21.0 ± 1.4	83.1 ± 1.1	19	84.3 ± 7.2
Vehicle	18	65.4 ± 3.5	9.4 ± 2.1	50.1 ± 7.9	11	59.4 ± 4.7
Votes	16	96.1 ± 1.5	8.2 ± 1.5	97.0 ±0.7	7	97.9 ± 1.3
Vowel	10	69.8 ± 6.4	6.8 ± 1.2	70.2 ±1.6	6	71.5 ± 5.7
Wine	13	97.1 ± 4.0	8.2 ± 1.2	96.7 ±0.7	7	97.1 ± 3.9
Zoo	16	96.0 ± 4.9	8.8 ± 1.6	96.8 ±2.0	9	99.0 ± 3.0
Abstract1	790	89.0±9.4	402.2 ± 14.2	89.2 ± 1.0	387	91.0 ± 9.4
Abstract2	790	84.0±12.0	389.8 ± 5.2	84.0 ± 1.1	382	85.0 ± 10.2
Reuters1	1568	91.6±2.9	766.0 ± 12.0	90.2 ± 0.7	750	91.5 ± 0.7
Reuters2	435	88.5±10.5	222.4 ± 14.7	90.3 ± 0.8	195	91.5 ± 10.6
Reuters3	1440	96.4±1.6	721.0 ± 16.6	96.2 ± 0.7	712	96.9 ± 1.6

in all of the datasets.

Table 6.4 compares the results of GADistAI with the results of other GA-based (ADHOC) [Richeldi & Lanzi, 1996] and several non GA-based approaches that are available in the literature (non-GA) [Liu & Setiono, 1996a; Liu & Setiono, 1996b; Kohavi, 1994; Kohavi & Frasca, 1994; Koller & Sahami, 1996; Koller & Sahami, 1997]. A '-' indicates that the result is not reported in the corresponding reference. The results indicate that GADistAI gave higher generalization accuracy than the other techniques or comparable accuracy in almost all cases (except **Vehicle** dataset) although it occasionally selected more features. GADistAI produced feature subsets with larger number of features than the approach in [Koller & Sahami, 1996; Koller & Sahami, 1997] for **Reuters** datasets. This can be explained by that the former found the feature subsets using a genetic algorithm for datasets with relatively large number of features while the latter set up the number of features to select a priori. It should be noted that it is not generally feasible to do a completely fair and thorough comparison between different approaches without the complete knowledge of the parameters and the set up used in the experiments.

6.5.2.2 Minimizing Cost and Maximizing Accuracy using Feature Subset Selection

The selection was based on both the generalization accuracy and the measurement cost of features. (See the fitness function in equation (6.1)). The 3-bit parity problem, Cleveland heart disease, hepatitis domain and pima indians diabetes datasets were used for the experiment (with the random costs in the 3-bit parity problem). The results are shown in Table 6.5 and 6.6 for randomly partitioned and 10-fold cross-validation datasets, respectively.

As we can see from Table 6.5, the fitness function that combined both accuracy and cost outperformed that based on accuracy alone in every respect: the number of features used, generalization accuracy, and the cost. This is not surprising because the former tries to minimize cost (while maximizing the accuracy), which reduces the number of features, while the latter emphasizes only on the accuracy.

Table 6.4 Comparison between various approaches for feature subset selection.

	non-GA		ADHOC		GADistAI	
Dataset	*Features*	*Accuracy*	*Features*	*Accuracy*	*Features*	*Accuracy*
Annealing	-	-	8	95.0	18	99.5
Cancer	4	74.7	-	-	8	98.6
CRX	6	85.0	7	85.1	6	88.0
Glass	4	62.5	4	70.5	5	71.0
Heart	3	79.2	5	80.8	7	85.9
Hepatitis	4	84.6	-	-	10	88.7
Horse	4	85.3	-	-	5	85.0
Pima	-	-	3	73.2	2	76.8
Sonar	-	-	16	76.0	28	85.5
Vehicle	-	-	7	69.6	11	59.4
Votes	4	97.0	5	95.7	7	97.9
Reuters1	40	94.1	-	-	750	91.5
Reuters2	40	90.0	-	-	195	91.5
Reuters3	80	98.6	-	-	712	96.9

Table 6.5 Comparison of different fitness evaluations for randomly partitioned datasets.

	Accuracy only			Accuracy & Cost		
Dataset	*Features*	*Accuracy*	*Cost*	*Features*	*Accuracy*	*Cost*
3P	6.6	100	46.1	4.3	100	26.7
HeartCle	7.3	92.9	335.7	6.1	93.0	261.5
Hepatitis	9.2	97.1	22.8	8.3	97.3	19.0
Pima	3.8	79.5	28.5	3.1	79.5	22.8

Table 6.6 Comparison of different fitness evaluations for 10-fold cross-validation.

	Accuracy only			Accuracy & Cost		
Dataset	*Features*	*Accuracy*	*Cost*	*Features*	*Accuracy*	*Cost*
3P	4.8	100	35.6	3.8	100	25.4
HeartCle	8.4	86.9	390.5	7.2	85.7	317.8
Hepatitis	10.2	85.2	23.4	10.0	85.3	23.2
Pima	4.0	73.1	29.3	4.2	76.1	20.8

Table 6.6 also shows the fitness function that combined both accuracy and cost outperforms that based on accuracy alone in all datasets except **HeartCle**. The generalization accuracy was higher and the cost was also higher with the fitness function that is based on accuracy alone in **HeartCle** dataset. This explains how the fitness function (equation (6.1)) works in GADistAl and verifies the rationale behind it. Also, note that some of the runs resulted in feature subsets which did not necessarily have minimum cost. This suggests the possibility of improving the results by the use of a more principled choice of a fitness function that combines accuracy and cost.

6.6 Summary and Discussion

An approach to feature subset selection using a genetic algorithm for neural network pattern classifiers is proposed in this paper. A fast inter-pattern distance-based constructive neural network algorithm, DistAl, is employed to evaluate the fitness (in terms of the generalization accuracy) of candidate feature subsets in the genetic algorithm. The results presented in this paper indicate that genetic algorithms offer an attractive approach to solving the feature subset selection problem (under a different cost and performance constraints) in inductive learning of pattern classifiers in general, and neural network pattern classifiers in particular.

The GA-based approach to feature subset selection does not rely on monotonicity assumptions that are used in traditional approaches to feature selection which often limits their applicability to real-world classification and knowledge acquisition tasks. It also offers a nat-

ural approach to feature subset selection by taking into account, the distribution of available data. This is due to the fact that feature selection is driven by estimated fitness values, which if based on multiple partitions of the dataset into training and test data, provide a robust measure of performance of the feature subset. This is not generally the case with many of the greedy stepwise algorithms that select features based on a single partition of the data into training and test sets. Consequently, the feature subsets selected by such algorithms are likely to perform rather poorly on other random partitions of the data into training and test sets.

The approach to feature subset selection is able to naturally incorporate multiple criteria (e.g., accuracy, cost) into the feature selection process. This finds applications in cost-sensitive design of classifiers for tasks such as medical diagnosis, computer vision, among others. Another interesting application is automated data mining and knowledge discovery from datasets with an abundance of irrelevant or redundant features. In such cases, identifying a relevant subset that adequately captures the regularities in the data can be particularly useful, particularly in scientific knowledge discovery tasks. Techniques similar to the one discussed in this paper have been successfully used recently to select feature subsets for pattern classification tasks that arise in power system security assessment [Zhou *et al.*, 1997], sensor subsets in the design of behavior and control structures for autonomous mobile robots [Balakrishnan & Honavar, 1996a; Balakrishnan & Honavar, 1996b].

Some avenues for future research include:

- Extensive experimental (and wherever feasible, theoretical) comparison of the performance of the proposed approach with that of conventional methods for feature subset selection is of interest and significance.

- Our approach, GADistAI can be applied to more real-world applications, and its effectiveness can be verified. For instance, GADistAI can be used in scientific knowledge discovery tasks in bioinformatics (e.g., discovery of protein structure–function relationships, carcinogenicity prediction, gene sequence identification).

- Different machine learning algorithms can be used within the genetic algorithm for feature

subset selection as our approach used a constructive neural network learning algorithm, DistAl. For instance, we incorporated the naive Bayesian classifier into a genetic algorithm and verified a much improved generalization accuracy.

- Feature subset selection can be extended by incorporating *feature construction* and *genetic programming* [Koza, 1992] as well.
- A simple fitness evaluation is used in our experiments to consider both the generalization accuracy and the measurement costs of attributes. More principled design of multi-objective fitness functions for feature subset selection using domain knowledge as well as mathematically well-founded tools of multi-attribute utility theory [Keeney & Raiffa, 1976].

7 CONSTRUCTIVE THEORY REFINEMENT IN KNOWLEDGE BASED NEURAL NETWORKS

7.1 Introduction

As described in chapter 1, *inductive learning* [Langley, 1995; Mitchell, 1997; Honavar *et al.*, 1999a] is an approach to learn concept descriptions (or knowledge) from a set of labeled examples. For instance, the various neural network learning algorithms described in chapter 2, 4, 5, and the decision tree algorithms [Quinlan, 1986; Quinlan, 1993] are inductive learning algorithms. Such inductive learning systems have yielded good generalization in many applications. However, the generalization is from the set of labeled examples not knowing how the examples are labeled. The performance of inductive learning systems can potentially improve by using domain specific knowledge or a domain theory about the concept being learned (when such domain knowledge/theory is available). Hybrid learning systems that incorporate domain knowledge in inductive learning can yield better performance (in terms of learning speed and generalization capability, for instance) than learning systems that rely on induction with the labeled data only. On the other hand, the domain specific knowledge is often incomplete or inaccurate in many applications of interest.

Inductive learning systems that use information from training examples to modify an existing domain theory by either augmenting it with new knowledge or by refining the existing knowledge are called *theory refinement* systems.

Theory refinement systems can be classified into the following three categories [Parekh, 1998]:

- *Symbolic approaches*

  Symbolic learning algorithms (e.g., decision tree learning algorithm [Quinlan, 1986; Quin-

lan, 1993]) are used for revising the theory in this category. For instance, the EITHER system [Ourston & Mooney, 1994] finds a subset of examples that the domain theory is not able to correctly classify, and generates new rules that correctly classify the previously misclassified subset of examples using the decision tree learning algorithm.

- *Inductive Logic Programming (ILP) based approaches*

  This approach uses computational logic as the knowledge representation mechanism and refine the theory using the inductive inference rather than the traditional deductive inference. For instance, the FORTE [Richards & Mooney, 1995] employs a hill-climbing search for refining theories represented by first-order Horn-clauses. It identifies errors in the theory and keep revising it using a library of operators until no further revisions are possible.

- *Connectionist approaches*

  Neural networks are good candidate to theory refinement. The domain knowledge can be embedded into an initial network architecture either by explicitly setting the values for the connection weights or by training them. Then it can be refined by training the network on a set of labeled examples. For instance, the KBANN [Towell *et al.*, 1990; Towell & Shavlik, 1994] determines an initial network architecture by translating *AND-OR* graph representation of the domain knowledge, and applies the backpropagation learning algorithm [Rumelhart *et al.*, 1986] to refine the knowledge. KBANN is reported to outperform symbolic approaches (such as EITHER) and other learning algorithms (such as backpropagation and ID3 [Towell & Shavlik, 1994; Parekh, 1998]. However, KBANN is limited in that the network topology is not allowed to change, and thus theory is refined in terms of weight modifications only. In addition, new set of rules can not be incorporated.

We attempt to use prior knowledge in neural network learning as it is demonstrated and claimed to be efficient in [Shavlik, 1994]. We incorporate the domain specific knowledge into the initial network and refine it using a neural network learning algorithm. As explained

in chapter 4, constructive neural network learning algorithms have potential for generating near-minimal networks commensurate with the task dynamically, and have advantages over backpropagation style learning algorithms. Constructive learning algorithms also offer an interesting approach to the use of (possibly inaccurate and incomplete) domain knowledge for refining it. The domain theory can be translated into an initial network architecture (either by explicitly setting the values for the connection weights or by training them), and any new rules can be easily incorporated into the network. Then, any inaccuracy or inconsistency in the domain theory can now be refined by adding neurons to the network and training them on a set of labeled examples. Figure 7.1 depicts this process.

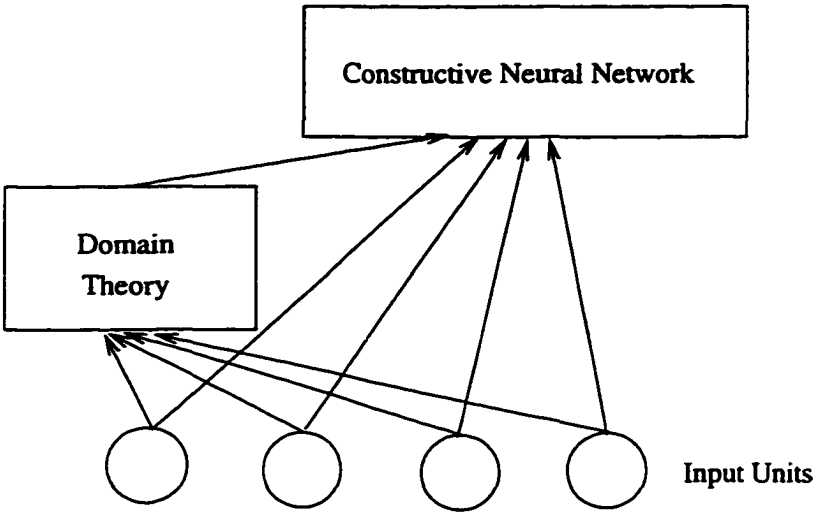


Figure 7.1 Constructive learning for theory refinement.

Against this background, this chapter describes a constructive approach to theory refinement using knowledge based neural networks. The rest of this chapter is organized as follows: Section 7.2 describes the process of incorporating the domain knowledge into an initial network architecture and refining it using DistAl. Section 7.3 summarizes several related constructive theory refinement systems to be compared with our approach. Section 7.4 presents the results of our experiments to compare our approach with DistAl with other constructive theory refinement systems described in section 7.3. Section 7.5 concludes with summary and discussion of

some directions for future research.

7.2 Theory Refinement Using DistAI

The domain theory is typically represented as a set of *rules*. The set of rules constitutes a hierarchy. Each rule in the hierarchy determines its output based on rules below it, and the rule at the top of the hierarchy produces an output, which is the final decision of the set of rules.

The rules can be translated into neural networks as suggested in [Towell *et al.*, 1990; Parekh, 1998]. The rules can be arranged in the form of having only one antecedent in each rule. Then the modified set of rules can be converted into an *AND-OR* graph, which can be incorporated in a neural network with perceptrons with proper weight settings. (See [Parekh, 1998] for detailed description and examples of rule incorporation into a neural network).

In our system, the set of rules are incorporated into neural networks as a blackbox. As shown in Figure 7.1, it computes an output using the rules inside and the inputs. This eliminates the need for converting the rules into an AND-OR graph, and then into a neural networks of perceptrons.

The domain theory is incorporated into the initial network architecture as shown in Figure 7.1 and refined using DistAI in our constructive theory refinement system. Both the inputs and an additional value out of the blackbox of domain theory are fed into DistAI, and the network is trained.

7.3 Previous Constructive Theory Refinement Systems

In this section, we will introduce several constructive theory refinement systems to make comparisons with our approach.

7.3.1 HDE

Fletcher and Obradović proposed a constructive theory refinement system [Fletcher & Obradović, 1993]. They start with an initial network representing the domain theory and refines

the theory by training a single hidden layer of perceptrons using a set of labeled examples. The *hyperplane detection* from *examples* algorithm [Baum & Lang, 1991] is used to generate the hidden neurons. The hyperplane detection from examples algorithm divides pattern space with hyperplanes. A hyperplane is randomly constructed by choosing two points that belong to different classes and localizing a split between them. This process is repeated until a fixed number of hyperplanes are generated. The hyperplanes are mapped to perceptrons in the hidden layer of the network, and the output unit is trained using the *pocket algorithm* [Gallant, 1990; Gallant, 1993]. We will name this Fletcher and Obradović's approach HDE.

7.3.2 TopGen

The TopGen algorithm [Opitz & Shavlik, 1995] searches the space of possible extensions of a KBANN network, and determines an extension with the best generalization accuracy on the cross-validation set. In other words, TopGen generates a KBANN using the domain theory, trains it using the backpropagation algorithm and puts it on a queue of candidate hypotheses. The algorithm chooses the best (in terms of classification accuracy) network from the queue and generates new networks by adding neurons at different locations in the best network chosen. These new networks are trained and placed on the queue. After this process is repeated for a number of iterations, the best network on the queue is returned.

7.3.3 REGENT

The REGENT algorithm [Opitz & Shavlik, 1997] extends TopGen by performing a genetic search in the space of network architectures. It starts with a population of networks from the initial KBANN. The fitness value of an individual network is the error on a cross-validation set. Two genetic operators, *crossover* and *mutation*, are specialized for connectionist theory refinement, and applied to the individuals in the population to generate new candidate networks. The crossover operator tries to maintain the rule structure of the network, and the mutation operator adds a node to the network using the TopGen algorithm. It is reported that both TopGen and REGENT outperform the backpropagation and KBANN algorithms.

7.3.4 MTiling-MPyramid

The domain knowledge (e.g., set of rules) is translated into a network of perceptrons as described in section 7.2. Among the various constructive neural network algorithms introduced in chapter 4, a combination of *MTiling* and *MPyramid* algorithms is used in this system [Parekh & Honavar, 1998]. The former is used to discretize the inputs for the latter, and the latter is used for the constructive theory refinement. In other words, this hybrid approach takes in real-valued patterns and produces a binary representation using the *MTiling* algorithm, and apply the binary representation to the *MPyramid* algorithm to refine the theory. Figure 7.2 depicts this hybrid system.

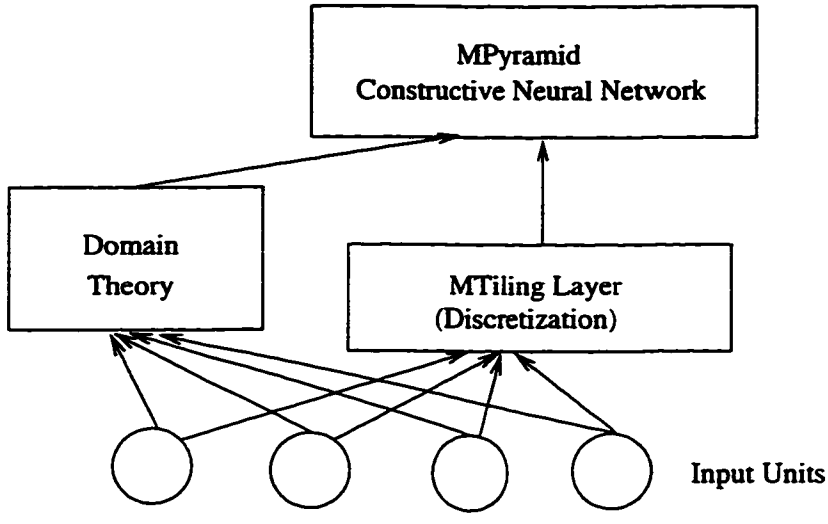


Figure 7.2 Hybrid constructive network for theory refinement.

7.4 Experiments

7.4.1 Datasets

Three different datasets were used in our experiments:

- **Ribosome**

  This data is from the Human Genome Project. It comprises of a domain theory and a

set of labeled examples. The input is a short segment of DNA nucleotides, and the goal is to learn to predict whether the DNA segments contain a ribosome binding site. There are 17 rules in the domain theory, and 1880 examples in the dataset.

- **Promoters**

  This data is also from the Human Genome Project, consists of a domain theory and a set of labeled examples. The input is a short segment of DNA nucleotides, and the goal is to learn to predict whether the DNA segments contain a promoter site. There are 31 rules in the domain theory, and 940 examples in the dataset.

- **financial advisor**

  The financial advisor rule base contains 9 rules as shown in Figure 7.3.

1	if (sav_adeq and inc_adeq)	then	invest_stocks
2	if dep_sav_adeq	then	sav_adeq
3	if assets_hi	then	sav_adeq
4	if (dep_inc_adeq and earn_steady)	then	inc_adeq
5	if debt_lo	then	inc_adeq
6	if (sav $\geq$ dep * 5000)	then	dep_sav_adeq
7	if (assets $\geq$ income * 10)	then	assets_hi
8	if (income $\geq$ 25000 + dep * 4000)	then	dep_inc_adeq
9	if (debt_pmt $<$ income * 0.3)	then	debt_lo

Figure 7.3 Financial advisor rule base.

As in [Fletcher & Obradović, 1993], a set of 5500 labeled examples that is consistent with the rule base are randomly generated. 500 examples are used for training and the remaining 5000 is used for testing.

7.4.2 Experimental Results

We compare the performance of our approach using DistAl with the approaches shown in Section 7.3 for each dataset.

7.4.2.1 Human Genome Project Datasets

A ten-fold cross validation is used in each approach. The average training and test accuracies of the rules in domain theory alone were 87.29 ± 0.22 and 87.29 ± 2.03 for **Ribosome** dataset and 77.45 ± 0.56 and 77.45 ± 5.01 for **Promoters** dataset, respectively. Table 7.1 and 7.2 shows the average generalization accuracy and the average network size (along with the standard deviations where available) for **Ribosome** and **Promoters** datasets. respectively.

Table 7.1 Results of **Ribosome** dataset.

	Test %	*Size*
Rules alone	87.3 ± 2.0	–
DistAl	86.3 ± 2.4	40.3 ± 1.3
MTiling-MPyramid	90.3 ± 1.8	23 ± 0.0
TopGen	90.9	42.1 ± 9.3
REGENT	91.8	70.1 ± 25.1

Table 7.2 Results of **Promoters** dataset.

	Test %	*Size*
Rules alone	77.5 ± 5.0	–
DistAl	93.0 ± 2.8	12.2 ± 1.0
MTiling-MPyramid	96.3 ± 1.8	34 ± 0.0
TopGen	94.8	40.2 ± 3.3
REGENT	95.8	74.9 ± 38.9

As we can see in Table 7.1 and 7.2, our approach yielded a reasonable performance. For **Ribosome** datset, it produced a lower generalization accuracy than other approaches. it also generated bigger networks than MTiling-MPyramid and REGENT. For **Promoters** dataset, it produced comparable generalization accuracy with smaller network size.

The time taken in our approach is significantly faster than other approaches. This is because DistAl is much faster (see in Chapter 5) than perceptron learning algorithms and the backpropagation algorithm. REGENT is even slower since it employs the genetic search.

7.4.2.2 Financial Advisor Rule Base

As explained earlier, 5500 patterns were generated randomly to satisfy the rules in Figure 7.3, of which 500 patterns were used for training and the remaining 5000 patterns were used for testing the network. In order to experiment the algorithms with several different incomplete domain theories, some of the rules were pruned with its antecedents in each experiment. For instance, if *sav_adeq* was selected as the pruning point, then the rules for *sav_adeq*, *dep_sav_adeq*, and *assets_hi* are eliminated from the rule base. In other words rules 2. 3, 6, and 7 are pruned. Further, rule 1 is modified to read "*if (inc_adeq) then invest_stocks*". Then the initial network is constructed from this modified rule base and augmented using constructive learning.

Our experiments follow those performed in [Fletcher & Obradović, 1993] and [Parekh & Honavar, 1998; Parekh, 1998]. As we can see in Table 7.3 and 7.4, our approach either outperformed other approaches or yielded comparable results. It output higher accuracies than other approaches in several cases, and it always produced compact networks. As discussed earlier, the speed advantage of our approach is significant.

Table 7.3 Results of financial advisor rule base (DistAl and MTiling-MPyramid).

Pruning point	DistAl		MTiling-MPyramid		Rules alone
	Test %	Size	Test %	Size	Test %
dep_sav_adeq	89.2	17	91.2 ± 1.7	28.2 ± 3.6	52.4
assets_hi	99.5	2	99.4 ± 0.2	10 ± 0.0	99.5
dep_inc_adeq	97.5	6	94.3 ± 1.5	21.0 ± 3.1	90.4
debt_lo	92.9	9	94.1 ± 2.0	22.1 ± 4.0	81.2
sav_adeq	98.4	9	90.8 ± 1.5	26.4 ± 3.3	87.6
inc_adeq	92.3	15	83.8 ± 2.2	32.7 ± 2.9	67.4

7.5 Summary and Discussion

Theory refinement techniques offer an attractive approach to exploiting available domain knowledge to enhance the performance of data-driven knowledge acquisition systems. Neural networks have been used extensively in theory refinement systems that have been proposed in

Table 7.4 Results of financial advisor rule base (HDE).

Pruning point	HDE		Rules alone
	Test %	Hidden Units	Test %
dep_sav_adeq	92.7	31	75.1
assets_hi	92.4	23	93.4
dep_inc_adeq	85.8	25	84.5
debt_lo	84.7	30	61.7
sav_adeq	92.2	19	90.9
inc_adeq	81.2	32	64.6

the literature. Most of such systems translate the domain theory into an initial neural network architecture and then train the network to refine the theory. The KBANN algorithm is demonstrated to outperform several other learning algorithms on some domains [Towell *et al.*, 1990; Towell & Shavlik, 1994]. However, a significant disadvantage of KBANN is its fixed network topology. TopGen and REGENT algorithms are proposed to eliminate this limitation and attempt to modify the network architecture. Experimental results demonstrate that TopGen and REGENT outperform KBANN on several applications [Opitz & Shavlik, 1995; Opitz & Shavlik, 1997]. The MTiling-MPyramid algorithm builds a network of perceptrons. Its performance, in terms of classification accuracies attained, as reported in [Parekh & Honavar, 1998], is comparable to that of REGENT and TopGen, but at significantly lower computational cost.

We proposed a connectionist theory refinement system based on DistAI and compared its performance with several other connectionist theory refinement systems. While many connectionist strategies translate the domain theory into an initial network architecture, our approach directly uses the set of rules to utilize the domain theory without any translation. More importantly, our approach is significantly faster other connectionist approaches compared because DistAI trains the network without iterative weight update procedures (e.g., perceptron learning, backpropagation algorithm) and/or the expensive genetic search. Experimental results demonstrate that our approach yields comparable generalization accuracies and favorable network size.

Some directions for future research include:

- Our current framework does not allow any changes to the original rules of the domain theory. In some scenarios it might be beneficial to allow the theory refinement system to explicitly modify the original rules. Design of a constructive learning scheme that allows for direct modification of the existing domain theory and also adds new knowledge rules is worth exploring in depth. In addition, considering uncertainty in the domain theory is of interest.

- Extraction of rules from the trained neural networks is an actively pursued area of research and finds direct applicability in data mining. We have not yet explored approaches for extracting the refined knowledge rules from the trained network. We conjecture that our method for training constructive learning algorithms would make the knowledge extraction task simpler. Our approach uses hyperspherical threshold neurons whose operation can more easily be translated into rules than the *sigmoid* neurons typically used in backpropagation type algorithms. Further, since the original rules are left uncorrupted in our approach, the comprehensibility of rules extracted from the trained network is likely to improve significantly. There is significant interest in the study of efficient techniques for knowledge extraction from trained neural networks. The interested reader is referred to [Towell & Shavlik, 1993; Fu, 1993; Craven, 1996] for additional details.

- Recent research has focused on the use of neural networks for cumulative multi-task learning [Thrun, 1995]. The goal here is to exploit the prior knowledge acquired while learning the earlier tasks to make the learning of the later (possibly more difficult) tasks easier. A constructive neural network learning algorithm is also good candidate for multi-task learning.

8 MULTI-AGENT SYSTEMS AND MOBILE AGENTS FOR INFORMATION RETRIEVAL AND KNOWLEDGE DISCOVERY

8.1 Introduction

As introduced in chapter 1, translating recent advances in our ability to gather, store, and analyze a wide variety of data on multiple, geographically distributed, heterogeneous data and knowledge sources into significant breakthrough in distributed problem-solving and decision-making in various applications presents the following problems [Honavar *et al.*, 1998]:

- The large volumes of data, the range of potentially relevant and useful complex inter-relationships that need to be discovered, and the diversity of data sources challenge approaches to *data mining* and *knowledge discovery*. Hence, tools are necessary to support data-driven knowledge acquisition and incremental theory refinement from multiple, heterogeneous, structured as well as semi-structured data and knowledge sources (including multiple types of sensor data, text, images, and the like). Machine learning is a practical approach to data mining and knowledge discovery. Among various machine learning algorithms, artificial neural networks, methods to improve their performance, and some applications are introduced and studied for this purpose in previous chapters (chapter 2, 3, 4, 5, 6, 7).

- Design of complex information systems in order to be feasible, often requires modular design which involves the decomposition of the overall task into more manageable subtasks. *Multi-agent systems* are appropriate tools for this purpose. A multi-agent system consists of multiple autonomous agents, each of which is responsible for a data source (e.g., independently managed database) or analysis capability (e.g., a knowledge discov-

ery tool). Modular design also lends itself to being adapted and extended for a broader class of knowledge network applications. There should be mechanisms for communication, coordination and control of collections of agents to ensure satisfactory operation of multi-agent systems.

- The data and knowledge sources are often geographically distributed. This calls for the use of *information assistants* or *software agents* for intelligent, selective, and context-sensitive data gathering and data assimilation prior to large scale data analysis. Hence, tools for monitoring different data sources and routing the appropriate information selectively to relevant sites or specific users are needed. Since the information of interest is user and context-dependent, such tools have to be customizable to specific users and information contexts. Furthermore, given the large volumes of data involved, it is desirable to perform as much analysis as feasible at the sites where the data is located and transmit only the results of analysis rather than flooding the network with data. Such *mobile* software agents are of significant interest that can transport themselves to appropriate sites, carry out the computation on site, and return with useful results.

This chapter gives more detailed descriptions on multi-agent systems and mobile agents, and introduces our approaches to those problems.

8.1.1 Multi-agent Systems

A multi-agent system consists of a set of agents that cooperate each other to accomplish certain tasks of interest. Multi-agent systems, because of their modularity, offer an attractive approach to the design of such systems that utilize multiple, heterogeneous, and often autonomous resources in decision support, scientific, distributed manufacturing, business, military command and control, and other applications. In such multi-agent systems, satisfactory completion of the tasks at hand depend critically on effective communication and coordination among the agents. In order to harvest the potential power of such systems in practical applications, it is essential that suitable mechanisms be devised to exercise adequate control over the behavior of such systems. In multi-agent systems, the notion of control suggests such func-

tions as coordination among agents, synchronization among multiple agents, activation and deactivation of individual agents or groups of agents, selection among agents, creation of new agents when needed, elimination of agents that are no longer needed, adaptation of individual agents and agent populations to changes in the environments or task demands, learning (both at the individual as well as group levels) from experience, and (at a much slower timescale) evolution of agent populations toward more desirable behaviors. Both natural systems (e.g., cells, brains, immune systems, evolution, groups, social organizations, economies, societies) and artificial systems (computers, multi-computers, computer networks, programs, factories) offer rich sources of examples of a wide variety of coordination and control mechanisms that can be beneficially incorporated into the design of complex information processing systems in general [Honavar & Uhr, 1990; Uhr, 1984]: coordination that emerges from interaction among large number of agents that exhibit relatively simple behaviors inspired by organizations such as the ant colonies [Hofstadter, 1979]; hierarchical control where the flow of control follows the structure of the hierarchy (e.g., in the military); coordination that emerges from interaction (including communication and negotiation) among self-interested agents as exemplified in the *contract net protocol (CNP)* [Smith, 1980; Sandholm, 1993; Sandholm, 1998] and related negotiation mechanisms [Rosenschein & Zlotkin, 1994] and distributed routing in large self-managing communication networks [Mikler *et al.*, 1996]; control that emerges from competition for resources under the influence of environmental rewards as exemplified by evolutionary processes modeled by genetic algorithms [Mitchell, 1996].

The *Knowledge Query and Manipulation Language (KQML)* [Finin *et al.*, 1997] and the *CNP* [Smith, 1980] are two candidates that have been proposed for multi-agent coordination. The *KQML* provides a set of *performatives* which define the communicative acts that can be performed by the agents. Agent *tell* facts to other agents, *evaluate* expressions, *subscribe* to services, and so on. The CNP provides a framework for negotiation using bidding and contracts among agents in a distributed problem solver [Smith, 1980]. In other words, agents *announce*, *bid*, *award* and *take awards* for certain tasks continuously to maximize the overall utility of the system. (See Section 8.2.1 for details).

The CNP can be applied to different kinds of applications that can be solved efficiently by a distributed problem solver. The CNP can be used to support interactive negotiations among multiple agents in a distributed problem solver. One such application involves information retrieval and knowledge discovery from heterogeneous, autonomous, and distributed data and knowledge sources. Section 8.2 describes our approach to coordination and control of multiple agents for information retrieval through inter-agent negotiation using the CNP.

8.1.2 Mobile Agents

Mobile agents are such agents that can move in a computer network from host to host as needed in accomplishing their tasks. Mobile agents offer a natural extension of the *remote programming* paradigm in several interesting ways. In particular, mobile agents are generally thought to be able to act with a certain degree of autonomy. That is, the agent is able to make intelligent decisions regarding its itinerary and modify it in a dynamic fashion in response to information that becomes available as it moves from one host to another. Multiple mobile agents, when equipped with the ability to communicate with each other, can negotiate, collaborate, and compete with each other as appropriate in the pursuit of their objectives [Rosenschein & Zlotkin, 1994], as described earlier. Mobile agents provide a potentially efficient framework for performing computation and analysis in a distributed fashion at sites where the relevant data is available, and transmitting only the results of analysis instead of expensive shipping of large volumes of data across the network. For example, in many data mining and knowledge discovery tasks, a mobile agent can visit multiple, geographically distributed data repositories and return with *knowledge* (e.g., in the form of a few concise rules) that captures the observed regularities in the data. Unlike *remote procedure calls* which require ongoing communication through a failsafe network from the time of initiation of the task until its completion, a major advantage of mobile agents is that ongoing interaction does not require ongoing communication [White, 1997]. Mobile agents provide an attractive paradigm for design and implementation of scalable, flexible, and extensible systems for selective information retrieval and knowledge discovery from multiple, geographically distributed, heterogeneous data sources. Also, mobile

agents can easily adjust their behavior as needed based on the status of the network. For instance, in case of network failure, it can wait until the network connection is restored, or it can find alternative routes.

Mobile agent infrastructures support the creation, deployment, and management of mobile software agents. There is considerable ongoing research on mobile agent infrastructures [White, 1997; Kiniry & Zimmerman, 1997; Wong *et al.*, 1999]. Most of them have a similar architecture consisting of at least three components: agent servers, agent interface, and agent brokers (service directory). Agent servers support basic agent migration mechanisms, authentication, and perhaps provide other services. Agent brokers provide addresses of agent servers and support mechanisms for uniquely naming agents and agent servers. Agent interface is used by application programs to create and interact with agents. The different mobile agent infrastructure proposals differ from each other in terms of detailed implementation (e.g., choice of agent transport mechanism, agent programming languages, and so on). For the experiments described in this chapter, we used ObjectSpace's Voyager [1] which is a platform-independent mobile agent infrastructure that is written completely in Java. (See Section 8.3.1 for details).

Nowadays, it is possible for individuals around the world to access a wide variety of information sources on the Internet. However, effective use of these information sources (e.g., documents, articles, electronic mail messages, news, and the like), requires fairly sophisticated tools for locating, classifying, and retrieving only those items that are of interest to a given user of a group of users. For instance, a researcher might be interested in selectively retrieving recently published papers related to his or her research from a variety of sources. Similarly, an individual might be interested in selectively retrieving and reading news articles. This presents us with a *document classification problem*. This is just one of many similar tasks that need to be automated in order for people to be able to make effective use of the emerging computing, communications, and information infrastructure.

Document retrieval has been the subject of study for several decades [Salton, 1982]. However, work on personalized document retrieval agents is relatively new. Some examples include

[1] [http://www.objectspace.com/Products/voyager1.htm]

WebWatcher [Joachims *et al.*, 1997], **Personal WebWatcher** [Mladenic, 1996], **Fab** [Balabanovic & Shoham, 1997] which learn user interests using user feedback and recommend and/or prefetch web pages; and software agents for mail handling and electronic news filtering [Maes, 1997]. We implemented simple mobile agents for information retrieval from multiple, possibly heterogeneous knowledge and data sources using **Voyager**. The mobile agents are also customizable to individuals to consider their interests.

Against this background, the rest of this chapter is organized as follows: Section 8.2 describes the CNP and its implementation, and presents the results of simulations designed to evaluate the performance of CNP in a multi-agent system for information retrieval. Section 8.3 focuses on the design and implementation of customizable intelligent mobile agents for document retrieval from distributed document collections. Section 8.4 concludes with a summary and discussion of some directions for future research.

8.2 Design of a Multi-Agent System using the Contract Net Protocol

In this section, we describe the design and implementation of a multi-agent system for information retrieval and knowledge discovery using the CNP.

8.2.1 Contract Net Protocol

The CNP was originally proposed as a tool for communication and control in a distributed problem solver [Smith, 1980]. Its use was demonstrated in a distributed sensing system in [Smith, 1980] and for a distributed delivery system in [Sandholm, 1993]. CNP provides a mechanism for agents to communicate and negotiate to solve a distributed problem via *contracts*. A contract is a set of tasks to be accomplished. Agents *announce* tasks that they need performed, make *bids* to perform tasks announced by other agents, *evaluate* the bids and *award* contracts. Each agent has a set of *capabilities* and *needs*. An agent's capabilities indicate the set of tasks that the agent can perform for other agents (perhaps at a certain cost). An agent's needs indicate the set of tasks that the agent needs to accomplish. The tasks that are part of an agent's needs candidates to be delegated to other agents: the tasks are announced to

other agents, the bids for the tasks are collected, the best bid (which yields the highest utility e.g., profit, quality of the service, etc.) is determined and then the tasks are awarded to the agent submitting the best bid. Under certain assumptions, it can be shown that in successive contracts each contract leads to a task allocation that is more beneficial to the entire society of agents that is involved in the negotiation process [Sandholm, 1997].

8.2.2 Implementation Details

Our CNP implementation is based on a modular, object-oriented design which contains several domain-independent modules as well as a few domain-dependent (application-specific) modules. This design lends itself to being adapted for use in a broad range of applications. The domain-independent modules include: *Announcer*, *Bidder*, *Awarder*, *Awardee* as well as certain auxiliary modules. They work closely with the *Local Optimizer* module which calls application-dependent functions. The domain-independent modules operate as follows:

- *Directory* provides assistance in locating agents. When an agent enters into the network, it registers with the directory.
- *Messenger* handles sending and receiving of messages between agents. It queries the directory to locate agents. It also manages the storage and organization of messages. Other modules use these stored messages.
- *Phase Controller* controls the orderly execution of different stages of the negotiation process by calling the announcer, bidder, awarder and awardee.
- *Announcer* sends announcements to an appropriate set of agents. Each message defines the task to be fulfilled and sets any limitations (e.g., the maximum price it is willing to pay, the time limit it will be waiting to receive bids, and so on). The local optimizer selects a task from among the current needs for making an announcement. Only one announcement message is sent per cycle.
- *Bidder* generates bids in response to announcements received. The process is completed in two stages: It first discards announcements that correspond to tasks that can not be

performed given the current capabilities of the agent or announcement that have expired. It then determines the appropriate bids by consulting the local optimizer.

- *Awarder* works with the local optimizer to process various bids (received within some time limit), generating awards based on some criteria (e.g., lowest cost, highest quality, etc.), and sends awards to appropriate agents. The agents whose bids are not selected are sent *loser* messages.

- *Awardee* accepts an award that is received, communicates with the local optimizer to update the state of the system (e.g., if it is supposed to supply some products, it decreases the quantity of products and increases the income/assets).

The *local optimizer* is a domain or application dependent module which incorporates domain-specific features to make decisions. Each phase of negotiation relies on certain functions performed by the local optimizer which ensures that each agent behaves rationally so as to maximize its utility.

The four phases of *Announcer*, *Bidder*, *Awarder* and *Awardee* are repeated until the negotiation reaches an equilibrium (i.e., no agent is in a position to improve its utility by delegating or accepting tasks to be performed) or an acceptable solution with anticipated utility is obtained.

The overall view of the multi-agent system, the pseudo-code that summarizes the CNP, and the structure of an agent are shown in Figure 8.1, Figure 8.2 and Figure 8.3, respectively.

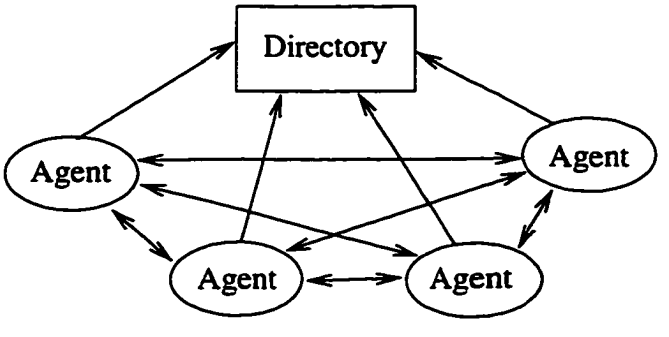


Figure 8.1 Overall view of the system.

do forever (in *Phase Controller*)

Bidder:

- Read all the announcements received and check their expiration times;
- Generate bids within the limit of resources available (e.g., cash) for announcements that has not expired;
- Send the bids to the corresponding agents;
- Delete announcements processed;

Awarder:

- Read all the bids for announcements received, and drop any bids that are late;
- Pick the best bids for announcements whose time limit is expired;
- Send an awarding message to the winning agent, and send a loser message to all the other agents who had bid;
- Delete all expired announcements;

Awardee:

- Accept all awarding messages; Update the state of the system;
- Get all the loser messages;
- Remove all the announcements received corresponding to the awarding/loser messages;

Announcer:

- Pick one item from *Needs* set (that gives highest utility) and send out announcements to agents;
- Re-announce announcements that did not receive any bids within the time limit;

end

Figure 8.2 The contract net protocol.

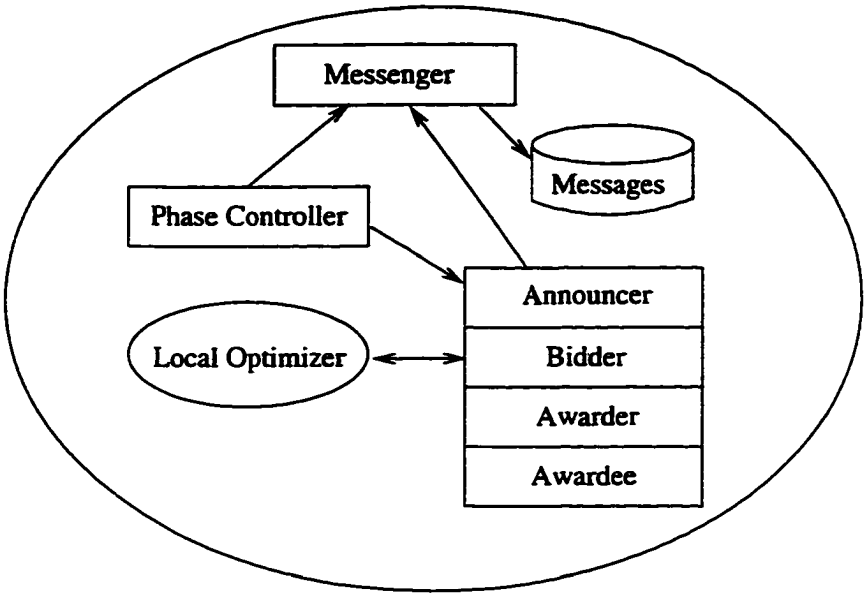


Figure 8.3 An agent using the contract net protocol.

8.2.3 Evaluation

In order to explore the feasibility of using CNP for coordination among agents in a distributed knowledge network, we designed and implemented a multi-agent system for information retrieval and knowledge discovery from distributed data sources. Each agent maintains a needs table and a capabilities table. The needs table contains description of the data (e.g. a list of queries) that the agent needs, and the capabilities table contains description of the agent's capabilities (e.g., the information that is able to supply, or analysis functions that is able to perform on data) that can be utilized by other agents. Items in both tables have costs associated with them. The cost in needs table is the maximum cost the agent is willing to pay to get the information. The cost in capabilities table represents the cost (measured in some suitable units) of accomplishing the task in question. These costs are used by the local optimizer during the negotiation process. Each agent seeks to maximize its utility. The framework is flexible enough to allow an agent to provide value-added information to other agents. For example, an agent can obtain data from other agents, perform analysis of the data to extract

some useful knowledge, and then provide the knowledge back to the agents that need it. Six agents were used in the CNP in our experiments.

The simulation results revealed the effectiveness of the CNP. Each agent was able to acquire the information that it needed from other agents at minimum cost attainable through the negotiation process. Sometimes, it was observed that some agents' bids were not sent in time (because of processing and communication delays) and thus were eliminated in the awarding phase. However, each agent was able to take advantage of the best bid among those that were available at any given time. More systematic experiments under a variety of conditions are needed to more completely characterize the performance of this system.

8.3 Intelligent Mobile Agents for Information Retrieval

In this section, we describe the design and implementation of intelligent mobile agents for retrieving relevant information using the Voyager mobile agent infrastructure.

8.3.1 The Voyager Mobile Agent Infrastructure

Several groups have designed and implemented mobile agent infrastructures. Most of them provide basic capabilities for creating, transporting, and managing mobile agents. An agent system, *Agent TCL*, is designed and implemented in [Gray *et al.*, 1996; Kotz *et al.*, 1997]. It includes network-sensing tools and a docking system for agents to move between computers. Three leading commercial mobile agent systems – General Magic's Odyssey [2], IBM's Aglets [3], and ObjectSpace's Voyager – are introduced and compared in [Kiniry & Zimmerman, 1997]. Recently, a set of standards for *Mobile Agent System Interoperability Facilities* (MAF) [FOKUS *et al.*, 1997] has been proposed by a consortium of several companies and research groups including the GMD FOKUS and IBM. MAF standardizes several key aspects of mobile agent infrastructure including: agent management, agent transfer, agent and agent system names, agent system types, and location (address) syntax. This will facilitate interoperability among different mobile agent systems that are based on different architecture, design, and implemen-

[2] [http://www.genmagic.com/technology/odyssey.html]
[3] [http://www.trl.ibm.co.jp/aglets]

tation choices. (See [Wong *et al.*, 1999] for a discussion of an MAF-compliant Java-based implementation of a mobile agent infrastructure).

Voyager is designed to help developers produce high impact distributed systems quickly. Voyager is implemented entirely in Java and designed to use the Java language object model. Voyager allows regular message syntax to construct remote objects, send them messages and move them between programs. The Voyager *Object Request Broker (ORB)* provides services for mobile objects and autonomous agents. It also provides services for mobile objects and autonomous agents. It also provides services for persistence, scalable group communication, and basic directory services.

Agent platforms like Odyssey or Aglets allow developers to create an agent, program it with a set of tasks, and launch it into a network to fulfill its mission. However, they have minimal support for basic distributed computing and treats agents differently than simple objects. Aglets use sockets and Odyssey uses *Remote Method Invocation (RMI)* to move agents between machines. However, none of these platforms allow sending a regular Java message to a stationary or moving agent. As a result, it is very difficult for objects to communicate with an agent after the agent has been launched and for agents to communicate with other agents.

An attractive feature of Voyager is that it seamlessly integrates distributed computing with agent technology. An agent in the Voyager system is a special kind of object that can move independently, can continue to execute as it moves, and otherwise behaves like any other Java object. Voyager enables objects and other agents to send standard Java messages to an agent even as the agent is moving. In addition, Voyager allows us to remote-enable any Java class, even a third party library class, without modifying the class source in any way. (See Voyager User Guide for further details).

8.3.2 Design of Customizable Document Retrieval Agents

We focus on the design of customizable agents for information retrieval from distributed data sources. We illustrate our approach to this problem using the customized document retrieval task. However, the proposed approach can be easily adapted to handle a wide range

of selective information retrieval tasks (e.g., image retrieval, DNA sequence retrieval, etc.).

Classification of documents necessarily has to involve some analysis of the contents of a document. In the absence of a satisfactory solution to the natural language understanding problem, most current approaches to document retrieval use a *bag of words* representation of documents [Salton, 1989; Korfhage, 1997]. A document is represented as a vector of weights for terms (or words) from a *vocabulary*. Although a variety of approaches can be used to design document classifiers using the bag of words representation, to keep the discussion focused, we restrict ourselves to a relatively simple yet effective approach based on the *TFIDF* (*term frequency - inverse document frequency*) classifier [Salton & McGill, 1983; Korfhage, 1997; Yang *et al.*, 1998d].

The TFIDF approach to document classification works as follows: Let $\mathcal{V}$ be the vocabulary used. Let d be a document. The document is processed using *stopping* and *stemming* procedures [Salton, 1989; Korfhage, 1997] to obtain a bag of words for document d. The stopping procedure eliminates all commonly used terms, and the stemming procedure [Porter, 1980] produces a list of representative (root) terms. Let w_i be the ith word in the vocabulary $\mathcal{V}$. The *term frequency* of w_i, $TF(w_i, d)$, is the number of times w_i occurs in d. The *document frequency* of w_i, $DF(w_i)$, is the number of documents in which w_i occurs at least once. The *inverse document frequency* of w_i, $IDF(w_i)$, is defined as

$$IDF(w_i) = log(\frac{|D|}{DF(w_i)})$$

where $|D|$ is the total number of documents. Then, the *term frequency - inverse document frequency* of w_i, $TFIDF(w_i, d)$ is given by [Salton, 1989; Korfhage, 1997]

$$TF(w_i, d) \cdot IDF(w_i)$$

The vector representation $\vec{d}$ of a document d is given by

$$\vec{d} = \left[TFIDF(w_1, d)\, TFIDF(w_2, d)\, \cdots TFIDF(w_{|\mathcal{V}|}, d)\right]$$

A TFIDF document classifier is constructed as follows: Let $\mathcal{C}$ be a collection of document classes of interest. A prototype vector $\vec{c}$ is generated for each class $c \in \mathcal{C}$ as follows:

$$\vec{c} = \sum_{d \in c} \vec{d}$$

A document n to be classified is assigned to the class whose prototype is the closest to it (as measured by some suitable distance function). The cosine function is a commonly used distance measure. Thus, the classification of n is given by

$$\arg\max_{c \in C} cos(\vec{n}, \vec{c})$$

$$= \arg\max_{c \in C} \frac{\vec{n} \cdot \vec{c}}{\| \vec{n} \| \cdot \| \vec{c} \|}$$

A simple version of the document classification task is to classify documents into two categories: interesting or uninteresting. This is accomplished by training a document classifier (i.e., constructing the class prototypes) using a set of pre-classified documents.

8.3.3 Implementation Details

The TFIDF classifier is incorporated into mobile agents on the Voyager mobile agent platform. First, a mobile agent (**Agent1**) is generated for searching and retrieving a set of documents from a remote site that matches with the query given by the user. The query is used to retrieve documents that match the query. The agent is shipped to the remote site, and the agent retrieves matching documents and sends them to the local site. Then the agent dies. The user then classifies the retrieved documents as interesting or not interesting. This provides a dataset for training the document classification and retrieval agent using the approach outlined above. Then, a TFIDF based document classification and retrieval agent is designed using the training data and the resulting agent (**Agent2**) is sent to a remote site to retrieve *relevant* documents. Relevant documents are determined by the classifier at the remote site and returned to the local site. Then the agent dies. Figure 8.4 depicts these agents.

8.3.4 Evaluation

The *paper abstracts* dataset (also shown in Table 6.1) was used in the experiments: 404 paper abstracts published between 1995 and 1997 were chosen from IEEE Expert magazine,

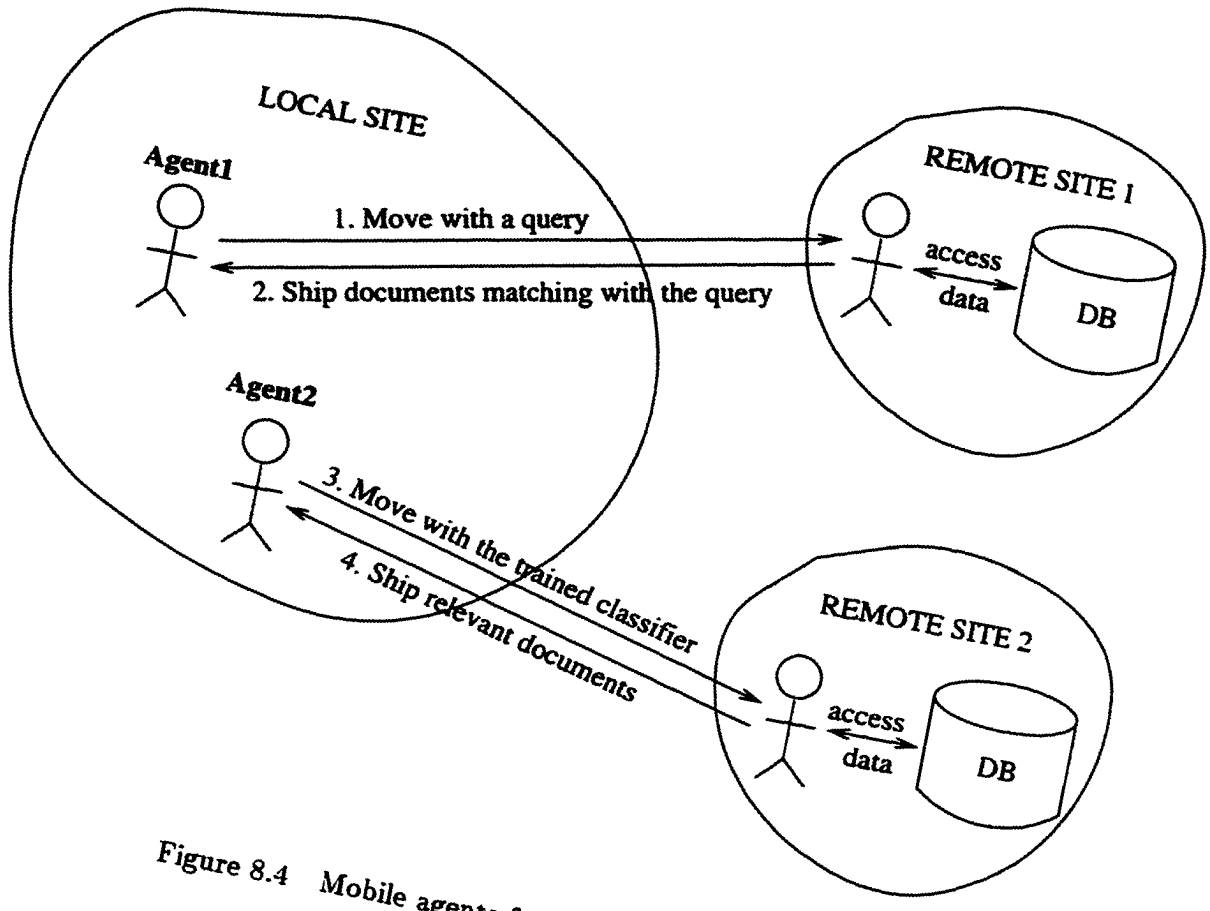


Figure 8.4 Mobile agents for document retrieval.

Journal of Artificial Intelligence Research and Neural Computation. Each abstract is represented as a feature vector as explained in section 8.3.2. A query was given to retrieve paper abstracts that is related to some topics of interest. The query vector is compared with each document vector to select 100 paper abstracts that match with the query. The selected paper abstracts were classified by two different users either interesting or not interesting to obtain two training sets (**Abstract1** and **Abstract2** in Table 6.1). There are 790 features and two classes in the datasets.

Instead of downloading all documents from the distributed databases, the agents worked on the remote databases, retrieved only a subset of relevant documents and sent them to the local site thereby minimizing the duration of the expensive network connection. In our experiments, the amount of data transferred in the mobile agents (the classifier and the relevant information) was much less than that in a conventional system (the entire data). The savings of network connection in mobile agents will be even greater for very large data while the mobile agents are lightweight.

8.4 Summary and Discussion

Translating recent advances in our ability to gather, store, and analyze a wide variety of data on multiple, geographically distributed, heterogeneous data and knowledge sources into significant breakthrough in distributed problem-solving and decision-making in various applications requires tools such as machine learning, multi agent systems, and mobile agents. Since some machine learning algorithms and related work are discussed in previous chapters, multi-agent systems and mobile agents are studied in this chapter. The former involves coordination and negotiation among agents to fulfill certain tasks, and the latter make agents move around the network from host to host and work on the available data and return the results to starting place instead of flooding the network with data. We implemented and evaluated a multi-agent system and mobile agents for information retrieval.

The CNP is used in the multi-agent system. CNP provides an attractive framework for negotiation and coordination among self-interested rational agents. Within this framework,

each agent can announce tasks, make bids, evaluate bids made by other agents to complete the tasks, and offer contracts. Hence, it offers a possible mechanism for coordination among multiple autonomous agents in multi-agent systems. Our implementation of the CNP, because of its modular design, lends itself to being adapted for use in a broad range of applications of distributed knowledge networks [Honavar *et al.*, 1998]. The feasibility of this approach to coordination among multiple autonomous agents was demonstrated on an application involving information retrieval from distributed data sources.

Intelligent mobile agents offer an attractive paradigm for the design of modular, flexible, robust, scalable, and adaptive information systems for a variety of applications, including customized information retrieval. Machine learning appears to be the most practical approach to designing customizable software agents. We have presented the design of mobile agents for customized document classification and retrieval using the commercially available Voyager mobile agent infrastructure. The experiments reported here demonstrate the effectiveness of machine learning as a viable and practical approach to the design of such agents. The performance of the agents in terms of classification accuracy on novel documents not used during training is quite good, and can be improved further using automated feature subset selection. Mobile agents offer a significant performance advantage over conventional remote procedure calls when we deal with very large remote data collections only a small fraction of which is of interest to the user.

Some interesting and promising directions for further research are outlined below:

- There are still many issues to be explored in multi-agent systems. For instance, the scalability of our approach in large multi-agent systems is of interest. Also, considering hybrid coordination mechanisms that use CNP in combination with a hierarchical organization of multi-agent systems is of interest.

- The CNP uses very simple languages to communicate between agents. The design of adequately expressive languages for representing the needs and capabilities of agents across different classes of applications is of interest.

- Multi-agent systems with the CNP can be used to solve other tasks. For instance, it can be applied to distributed manufacturing system where each agent has lists of parts to be needed or to be supplied. Agents can negotiate to maximize their profits and products.
- The CNP can be extended to consider *subcontracts*. Each agent can try to find the best deal through subcontracts.
- Systematic experiments with distributed data sources, under a variety of network conditions and data source characteristics (e.g., percentage of relevant documents) would be helpful in evaluating the scalability of this approach to real-world applications.
- Systematic studies to explore the relative performance advantages and disadvantages of alternative designs and implementations of such systems (including the mix of mobile and static agents) is of interest.
- The design of customizable information retrieval agents for a broad range of semi-structured and unstructured data sources (e.g., genome data, image data, etc.) is also of interest.

9 CONCLUSION

We have explored several related research problems in the design of systems of intelligent learning agents for information retrieval and data-driven knowledge discovery. These problems are of significant interest since they offer sophisticated tools for locating, classifying, selectively retrieving, and extracting knowledge from a wide variety of heterogeneous, distributed information sources accessible through the Internet. The major contributions of this dissertation include:

- **Multi-category variants of perceptron learning algorithms for non-separable classification tasks**

  Three different perceptron learning algorithms (the *pocket algorithm with ratchet modification*, the *thermal perceptron* and the *barycentric correction procedure*) were extended for multi-category classification. The performances of the algorithms (in both winner-take-all and independent training) were compared with carefully chosen artificial and real-world datasets in order to explore the inductive bias of each algorithm.

- **Multi-cagetory extension of constructive neural network learning algorithms**

  Many of the constructive neural network learning algorithms proposed in the literature are restricted in the number of output classes that they can classify, and the feature types that they can handle. We have extended these algorithms to solve problems with multi-category, real-valued patterns. Our extension guarantees 100% classification accuracy on finite, non-contradictory datasets. The effectiveness of these algorithms were demonstrated on a variety of real-world data [Parekh, 1998].

- **Design of an efficient inter-pattern distance-based constructive neural network learning algorithm**

  A new, provably convergent constructive neural network learning algorithm, DistAl, was designed and implemented. DistAl does not require time-consuming, iterative weight update procedures. It makes use of inter-pattern distance which can be computed by one pass through of the dataset. Therefore, it generates a network for a given task significantly faster. In addition, DistAl can handle multi-category classification tasks with a variety of features (e.g., binary, real-valued, nominal, etc.). Our experimental results on various artificial as well as real-world datasets demonstrate the high speed learning of DistAl, and present comparable classification accuracy to other sophisticated algorithms that rely on iterative weight updates. Because of its relatively high speed, DistAl is a good candidate for largescale data mining and knowledge discovery.

- **Feature subset selection with DistAl using a genetic algorithm**

  As DistAl is a variant of instance-based learning, the performance of the algorithm depends heavily on the values of data features. Classifying an appropriate set of features can thus improve the classification accuracy over the entire set of features. Each feature might entail a certain measurement cost. Therefore, feature subset selection decreases the overall cost of measurement of features. In our experiments, a genetic algorithm is used for DistAl to find an appropriate feature subset for better performance. Experimental results demonstrate that feature subset selection improves the classification accuracy of DistAl and decreases the number of features and measurement costs.

- **Knowledge-based theory refinement using DistAl**

  The availability of (often incomplete or inaccurate) domain specific knowledge about the concept being learned can potentially enhance the performance of the inductive learning system. Constructive neural networks offer a natural framework for knowledge-based theory refinement by incorporating the domain knowledge into the initial network architecture and then refining it using a learning algorithm. We used DistAl for theory refinement and compared its performance with other systems on several interesting ap-

plications with domain knowledge. The experiments revealed comparable performance to other approaches despite the simplicity and high speed of DistAI.

- **Multi-agent systems for information retrieval and knowledge discovery**
  Multi-agents systems provide an attractive approach to knowledge discovery from distributed data and knowledge sources. Mobile agents are particularly attractive since they can move in the network, work at the remote sites, and transmit the results only instead of wasting the network connection to ship the raw data across the network. We designed a system of mobile information retrieval agents using the contract net protocol as a framework for negotiations among agents. The effectiveness of our system was verified in the information retrieval and knowledge discovery task.

All of the research problems considered in this dissertation address important elements in designing a modular and scalable system for information retrieval and knowledge discovery from the abundant distributed, heterogeneous data and knowledge sources. Several future research problems are outlined in previous chapters. Some of the key future research directions include: alternative architectures for intelligent agents; techniques for interoperability in heterogeneous data and k- ıedge sources (e.g., multidatabase systems [Sheth & Larson, 1990]. mediator based systems [Wiederhold, 1997]); alternative approaches for multi-agent coordination and control; efficient algorithms for incremental knowledge discovery from knowledge or dynamic data collections.

BIBLIOGRAPHY

Aha, D. (1991). Incremental Constructive Induction: An Instance-based Approach. Pages 117–121 of: *Proceedings of the Eighth International Workshop on Machine Learning.* San Mateo, CA: Morgan Kaufmann.

Aha, D., Kibler, D., & Albert, M. (1991). Instance-Based Learning Algorithms. *Machine Learning*, **6**, 37–66.

Almuallim, H., & Dietterich, T. (1994). Learning Boolean Concepts in the Presence of Many Irrelevant Features. *Artificial Intelligence*, **69**(1-2), 279–305.

Andersen, H., & Tsoi, A. (1993). A Constructive Algorithm for the Training of a Multilayer Perceptron Based on the Genetic Algorithm. *Complex Systems*, **7**, 249–268.

Andersen, T., & Martinez, T. (1996). The Effect of Decision Surface Fitness on Dynamic Multi-layer Perceptron Networks (DMP1). Pages 177–181 of: *Proceedings of the World Congress on Neural Networks.*

Anlauf, J., & Biehl, M. (1990). Properties of an Adaptive Perceptron Algorithm. Pages 153–156 of: *Parallel Processing in Neural Systems and Computers.* Amsterdam, The Netherlands: Elsevier Science Pub. Co.

Ash, T. (1989). *Dynamic Node Creation in Backpropagation Networks.* Tech. rept. C-015. Institute for Cognitive Science, University of California, San Diego.

Balabanovic, M., & Shoham, Y. (1997). Fab: Content-Based, Collaborative Recommendation. *Communications of the ACM*, **40**(3).

Balakrishnan, K., & Honavar, V. (1995). Properties of Genetic Representations of Neural Architectures. Pages 807–813 of: *Proceedings of the World Congress on Neural Networks (WCNN-95)*, vol. 1.

Balakrishnan, K., & Honavar, V. (1996a). On Sensor Evolution in Robotics. Pages 455–460 of: *Proceedings of the 1996 Genetic Programming Conference.* MIT Press, Cambridge, MA.

Balakrishnan, K., & Honavar, V. (1996b). Some Experiments in the Evolutionary Synthesis of Robotic Neurocontrollers. Pages 1035–1040 of: *Proceedings of the World Congress on Neural Networks (WCNN-96).*

Banzaf, W., Nordin, P., Keller, R., & Francone, F. (1997). *Genetic Programming - An Introduction.* San Mateo, CA: Morgan Kaufmann.

Batchelor, B. (1978). *Pattern Recognition: Ideas in Practice.* New York: Plenum Press.

Baum, E., & Lang, K. (1991). Constructing Hidden Units using Examples and Queries. Pages 904–910 of: Lippmann, R., Moody, J., & Touretzky, D. (eds), *Advances in Neural Information Processing Systems, vol. 3.* San Mateo, CA: Morgan Kaufmann.

Bose, N., & Garga, A. (1993). Neural Network Design Using Voronoi Diagrams. *IEEE Transactions on Neural Networks*, 4(5), 778–787.

Brill, F., Brown, D., & Martin, W. (1992). Fast Genetic Selection of Features for Neural Network Classifiers. *IEEE Transactions on Neural Networks*, 3(2), 324–328.

Broomhead, D., & Lowe, D. (1988). Multivariable Functional Interpolation and Adaptive Networks. *Complex Systems*, 2, 321–355.

Burgess, N. (1994). A Constructive Algorithm That Converges for Real-Valued Input Patterns. *International Journal of Neural Systems*, 5(1), 59–66.

Campbell, C., & Vicente, C. (1995). The Target Switch Algorithm: A Constructive Learning Procedure for Feed-Forward Neural Networks. *Neural Computation*, 7, 1245–1264.

Carpenter, G., & Grossberg, S. (1991). *Pattern Recognition by Self-Organizing Neural Networks.* Cambridge, MA: MIT Press.

Caruana, R. (1993). Multitask Learning: A Knowledge-Based Source of Inductive Bias. Pages 41–48 of: *Proceedings of the Tenth International Conference on Machine Learning.* San Mateo, CA: Morgan Kaufmann.

Caruana, R., & Freitag, D. (1994). Greedy Attribute Selection. Pages 28–36 of: *Proceedings of the Eleventh International Conference on Machine Learning.* San Mateo. CA: Morgan Kaufmann.

Chen, C-H., Parekh, R., Yang, J., Balakrishnan, K., & Honavar, V. (1995). Analysis of Decision Boundaries generated by Constructive Neural Network Learning Algorithms. Pages 628–635 of: *Proceedings of WCNN-95, July 17-21, Washington D.C.*, vol. 1.

Church, A. (1941). The Calculi of Lambda Conversion. *Annals of Mathematics Studies.* **6.**

Clark, P., & Niblett, R. (1989). The CN2 Induction Algorithm. *Machine Learning*, **3**, 261–284.

Cormen, T., Leiserson, C., & Rivest, R. (1990). *Introduction to Algorithms.* Cambridge, MA: MIT Press.

Cost, S., & Salzberg, S. (1993). A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features. *Machine Learning*, **10**(1), 57–78.

Cover, T., & Hart, P. (1967). Nearest Neighbor Pattern Classification. *IEEE Transactions on Information Theory*, **13**, 21–27.

Craven, M. (1996). *Extracting Comprehensible Models from Trained Neural Networks.* Ph.D. thesis, Department of Computer Science, University of Wisconsin, Madison. WI.

Dasarathy, B. (1991). *Nearest Neighbor (NN) Norms: NN Pattern Classification Techiniques.* Los Alamitos, CA: IEEE Computer Society Press.

Dash, M., & Liu, H. (1997). Feature Selection for Classification. *Intelligent Data Analysis*, **1**(3).

Dayhoff, J. (1990). *Neural Network Architectures: An Introduction.* New York: Van Nostrand Reinhold.

Deffuant, G. (1995). An Algorithm for Building Regularized Piecewise Linear Discrimination Surfaces: The Perceptron Membrane. *Neural Computation*, **7**, 380–398.

Devijver, P. (1982). *Pattern Recognition: A Statistical Approach.* Englewood Cliffs, NJ: Prentice Hall.

Diday, E. (1974). Recent Progress in Distance and Similarity Measures in Pattern Recognition. Pages 534–539 of: *Proceedings of the Second International Joint Conference on Pattern Recognition.*

Doak, J. (1992). *An Evaluation of Feature Selection Methods and Their Application to Computer Security.* Tech. rept. CSE-92-18. Department of Computer Science, University of California, Davis, CA.

Domingos, P. (1995). Rule Induction and Instance-Based Learning: A Unified Approach. Pages 1226–1232 of: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-95).*

Duda, R., & Hart, P. (1973). *Pattern Classification and Scene Analysis.* New York: Wiley.

Fahlman, S. (1988). Faster-learning Variations on Backpropagation: an Empirical Study. Pages 38–51 of: Touretzky, D., Hinton, G., & Sejnowsky, T. (eds), *Proceedings of the 1988 Connectionist Models Summer School.* San Mateo, CA: Morgan-Kauffman.

Fahlman, S., & Lebiere, C. (1990). The Cascade Correlation Learning Algorithm. Pages 524–532 of: Touretzky, D. (ed), *Neural Information Systems 2.* San Mateo, CA: Morgan-Kauffman.

Finin, T., Labrou, Y., & Mayfield, J. (1997). KQML As an Agent Communication Language. In: Bradshaw, J. (ed), *Software Agents.* Cambridge, MA: MIT Press.

Fletcher, J., & Obradović, Z. (1993). Combining Prior Symbolic Knowledge and Constructive Neural Netwo rk Learning. *Connection Science*, **5**(3,4), 365–375.

Fogel, D. (1995). *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. Piscataway, NJ: IEEE Press.

FOKUS, GMD, IBM, & *et al.* (1997). *Mobile Agent System Interoperability Facilities Specification*. OMG TC Document orbos/97-10-05.

Foroutan, I., & Sklansky, J. (1987). Feature Selection for Automatic Classification of non-Gaussian Data. *IEEE Transactions on Systems, Man and Cybernetics*, **17**, 187–198.

Frean, M. (1990). The Upstart Algorithm: A Method for Constructing and Training Feedforward Neural Networks. *Neural Computation*, **2**, 198–209.

Frean, M. (1992). A Thermal Perceptron Learning Rule. *Neural Computation*, 4, 946–957.

Fu, L. M. (1993). Knowledge Based Connectionism for Revising Domain Theories. *IEEE Transactions on Systems, Man, and Cybernetics*, **23**(1), 173–182.

Fukunaga, K. (1990). *Introduction to Statistical Pattern Recognition*. New York: Academic Press.

Gallant, S. (1990). Perceptron Based Learning Algorithms. *IEEE Transactions on Neural Networks*, **1**(2), 179–191.

Gallant, S. (1993). *Neural Network Learning and Expert Systems*. Cambridge, MA: MIT Press.

Girosi, F., Jones, M., & Poggio, T. (1995). Regularization Theory and Neural Networks Architectures. *Neural Computation*, **7**, 219–269.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. New York: Addison-Wesley.

Golea, M., & Marchand, M. (1990). A Growth Algorithm for Neural Network Decision Trees. *Europhysics Letters*, **12**(3), 205–210.

Gray, R., Kotz, D., Nog, S., Rus, D., & Cybenko, G. (1996). *Mobile Agents for Mobile Computing.* Tech. rept. PCS-TR96-285. Dartmouth College, Hanover, NH.

Guo, Z. (1992). *Nuclear Power Plant Fault Diagnostics and Thermal Performance Studies Using Neural Networks and Genetic Algorithms.* Ph.D. thesis, University of Tennessee. Knoxville, TN.

Guo, Z., & Uhrig, R. (1992). Using Genetic Algorithms to Select Inputs for Neural Networks. Pages 223–234 of: *Proceedings of COGANN-92.*

Hanson, S., & Pratt, L. (1991). Comparing Biases for Minimal Network Construction with Back-propagation. Pages 177–185 of: Lippmann, R., Moody, J., & Touretzky, D. (eds). *Advances in Neural Information Processing Systems.* San Mateo, CA: Morgan Kaufmann.

Hassoun, M. (1995). *Fundamentals of Artificial Neural Networks.* Boston, MA: MIT Press.

Haykin, S. (1994). *Neural Networks: A Comprehensive Foundation.* New York: Macmillan.

Hebb, D. (1949). *The Organization of Behavior.* New York: John Wiley.

Hirose, Y., Yamashita, K., & Hijiya, S. (1991). Back-Propagation Algorithm Which Varies the Number of Hidden Units. *Neural Networks*, 4, 61–66.

Hofstadter, D. (1979). *Godel, Escher, Bach : an Eternal Golden Braid.* New York: Basic Books.

Holland, J. (1992). *Adaptation in Natural and Artificial Systems.* Cambridge, MA: MIT Press.

Honavar, V. (1990). *Generative Learning Structures and Processes for Generalized Connectionist Networks.* Ph.D. thesis, University of Wisconsin, Madison.

Honavar, V. (1994). Toward Learning Systems That Integrate Multiple Strategies and Representations. Pages 615–644 of: Honavar, V., & Uhr, L. (eds), *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration.* New York: Academic Press.

Honavar, V. (1999). Intelligent Agents. In: Williams, J., & Sochats, K. (eds), *Encyclopedia of Information Technology.* New York: Marcel Dekker. To appear.

Honavar, V., & Uhr, L. (1990). Coordination and Control Structures and Processes: Possibilities for Connectionist Networks. *Journal of Experimental and Theoretical Artificial Intelligence*, **2**, 277–302.

Honavar, V., & Uhr, L. (1993). Generative Learning Structures for Generalized Connectionist Networks. *Information Sciences*, **70**(1-2), 75–108.

Honavar, V., Miller, L., & Wong, J. (1998). Distributed Knowledge Networks. Pages 87–90 of: *Proceedings of IEEE Information Technology Conference.*

Honavar, V., Parekh, R., & Yang, J. (1999a). Machine Learning: Principles and Applications. In: Webster, J. (ed), *Encyclopedia of Electrical and Electronics Engineering.* New York: Wiley. In press.

Honavar, V., Yang, J., & Parekh, R. (1999b). Structural Learning. In: Webster, J. (ed), *Encyclopedia of Electrical and Electronics Engineering.* New York: Wiley. In press.

Hrycej, T. (1992). *Modular Learning in Neural Networks.* New York: Wiley.

Joachims, T., Freitag, D., & Mitchell, T. (1997). WebWatcher: A Tour Guide for the World Wide Web. Pages 770–777 of: *Proceedings of International Joint Conference on Artificial Intelligence.*

John, G., Kohavi, R., & Pfleger, K. (1994). Irrelevant Features and the Subset Selection Problem. Pages 121–129 of: *Proceedings of the Eleventh International Conference on Machine Learning.* San Mateo, CA: Morgan Kaufmann.

Keeney, R., & Raiffa, H. (1976). *Decisions with Multiple Objectives: Preferences and Value Tradeoffs.* New York: Wiley.

Kiniry, J., & Zimmerman, D. (1997). A Hands-on Look at Java Mobile Agents. *IEEE Internet Computing*, **1**(4), 21–30.

Kira, K., & Rendell, L. (1992). A Practical Approach to Feature Selection. Pages 249–256 of: *Proceedings of the Ninth International Conference on Machine Learning.* San Mateo, CA: Morgan Kaufmann.

Kitano, H. (1990). Designing Neural Networks Using Genetic Algorithms with Graph Generation System. *Complex Systems*, 4, 461–476.

Kohavi, R. (1994). Feature Subset Selection as Search with Probabilistic Estimates. In: *AAAI Fall Symposium on Relevance.*

Kohavi, R., & Frasca, B. (1994). Useful Feature Subsets and Rough Set Reducts. In: *Third International Workshop on Rough Sets and Soft Computing.*

Koller, D., & Sahami, M. (1996). Toward Optimal Feature Selection. Pages 284–292 of: *Proceedings of the Thirteenth International Conference on Machine Learning.* San Mateo, CA: Morgan Kaufmann.

Koller, D., & Sahami, M. (1997). Hierarchically Classifying Documents Using Very Few Words. Pages 170–178 of: *Proceedings of International Conference on Machine Learning.*

Kolodner, J. (1993). *Case-Based Reasoning.* San Mateo, CA: Morgan Kaufmann.

Kononenko, I. (1994). Estimating Attributes: Analysis and Extension of RELIEF. Pages 171–182 of: *Proceedings of European Conference on Mahcine Learning.*

Korfhage, R. (1997). *Information Storage and Retrieval.* New York: Wiley.

Kothari, R., & Agyepong, K. (1996). On Lateral Connections in Feed-forward Neural Networks. Pages 13–18 of: *Proceedings of International Conference on Neural Networks.*

Kotz, D., Gray, R., Nog, S., Rus, D., Chawla, S., & Cybenko, G. (1997). AGENT TCL: Targeting the Needs of Mobile Computers. *IEEE Internet Computing*, 1(4), 58–67.

Koza, J. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge, MA: MIT Press.

Krauth, W., & Mézard, M. (1987). Learning Algorithms with Optimal Stability in Neural Networks. *J. Phys. A: Math. Gen.*, **20**, L745–L752.

Kung, S. Y. (1993). *Digital Neural Networks.* Englewood Cliffs, NJ: Prentice-Hall.

Langley, P. (1994). Selection of Relevant Features in Machine Learning. Pages 1–5 of: *Proceedings of the AAAI Fall Symposium on Relevance.* Boston, MA: AAAI Press.

Langley, P. (1995). *Elements of Machine Learning.* San Mateo, CA: Morgan Kaufmann.

Le Cun, Y., Denker, J., & Solla, S. (1990). Optimal Brain Damage. Pages 598–605 of: Lippmann, R., Moody, J., & Touretzky, D. (eds), *Advances in Neural Information Processing Systems.* San Mateo, CA: Morgan Kaufmann.

Liu, H., & Setiono, R. (1996a). Feature Selection and Classification - A Probabilistic Wrapper Approach. Pages 419–424 of: *Proceedings of the Ninth International Conference on Industrial and Engineering Applications of AI and ES.*

Liu, H., & Setiono, R. (1996b). A Probabilistic Approach to Feature Selection - A Filter Solution. Pages 319–327 of: *Proceedings of the Thirteenth International Conference on Machine Learning.* San Mateo, CA: Morgan Kaufmann.

Lowe, D. (1995). Similarity Metric Learning for a Variable-Kernel Classifier. *Neural Computation*, **7**, 72–85.

Maes, P. (1997). Agents that Reduce Work and Information Overload. In: Bradshaw, J. (ed), *Software Agents.* Cambridge, MA: MIT Press.

Marchand, M., Golea, M., & Rujan, P. (1990). A Convergence Theorem for Sequential Learning in Two-Layer Perceptrons. *Europhysics Letters*, **11**(6), 487–492.

McCulloch, W., & Pitts, W. (1943). A Logical Calculus of Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics*, **5**, 115–133.

Mehrotra, K., Mohan, C., & Ranka, S. (1997). *Elements of Artificial Neural Networks.* Cambridge, Massachusetts: MIT Press.

Mézard, M., & Nadal, J. (1989). Learning Feed-forward Networks: The Tiling Algorithm. *J. Phys. A: Math. Gen.*, **22**, 2191–2203.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs.* Third edn. New York: Springer-Verlag.

Michalski, R., Mozetic, I., Hong, J., & Lavrac, H. (1986). The Multi-purpose Incremental Learning System AQ15 and Its Testing Application to Three Medical Domains. Pages 1041–1045 of: *Proceedings of the Fifth National Conference on AI.* Morgan Kaufmann.

Mikler, A., Honavar, V., & Wong, J. (1996). Analysis of Utility-Theoretic Heuristics for Intelligent Adaptive Network Routing. Pages 96–101 of: *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, vol. 1. Boston, MA: AAAI Press.

Minsky, M., & Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry.* Cambridge, MA: MIT Press.

Mitchell, M. (1996). *An Introduction to Genetic algorithms.* Cambridge, MA: MIT Press.

Mitchell, T. (1997). *Machine Learning.* New York: McGraw Hill.

Mladenic, D. (1996). *Personal WebWatcher: Design and Implementation.* Tech. rept. J. Stefan Institute, Ljubljana, Slovenia.

Modrzejewski, M. (1993). Feature Selection Using Rough Sets Theory. Pages 213–226 of: *Proceedings of the European Conference on Machine Learning.* New York: Springer.

Motwani, R., & Raghavan, P. (1996). Randomized Algorithms. *ACM Computing Surveys*, **28**(1), 33–37.

Mozer, M., & Smolensky, P. (1989). Skeletonization: A Technique for Trimming the Fat from a Network via Relevance Assessment. Pages 107–116 of: *Advances in Neural Information Processing Systems.*

Mucciardi, A., & Gose, E. (1971). A Comparison of Seven Techniques for Choosing Subsets of Pattern Recognition. *IEEE Transactions on Computers*, **20**, 1023–1031.

Murphy, P., & Aha, D. (1994). *Repository of Machine Learning Databases*. Department of Information and Computer Science, University of California, Irvine, CA.

Nadal, J. (1989). Study of a Growth Algorithm for a Feedforward Network. *International Journal of Neural Systems*, 1(1), 55–59.

Narendra, P., & Fukunaga, K. (1977). A Branch and Bound Algorithm for Feature Subset Selection. *IEEE Transactions on Computers*, **26**, 917–922.

Nilsson, N. (1965). *The Mathematical Foundations of Learning Machines*. New York: McGraw-Hill.

Nosofsky, R. (1986). Attention, Similarity, and the Identification-Categorization Relationship. *Journal of Experimental Psychology: General*, **115**, 39–57.

Opitz, D. W., & Shavlik, J. W. (1995). Dynamically Adding Symbolically Meaningful Nodes to Knowledge-Based Neural Networks. *Knowledge-Based Systems*, 8(6), 301–311.

Opitz, D. W., & Shavlik, J. W. (1997). Connectionist Theory Refinement: Genetically Searching the Space of Network Topologies. *Journal of Artificial Intelligence Research*, **6**, 177–209.

Ourston, D., & Mooney, R. J. (1994). Theory Refinement: Combining Analytical and Empirical Methods. *Artificial Intelligence*, **66**, 273–310.

Parekh, R. (1998). *Constructive learning: Inducing grammars and neural networks*. Ph.D. thesis, Iowa State University, Ames, IA.

Parekh, R., & Honavar, V. (1998). Constructive Theory Refinement in Knowledge Based Neural Networks. Pages 2318–2323 of: *Proceedings of the International Joint Conference on Neural Networks*.

Parekh, R., Yang, J., & Honavar, V. (1997a). *Constructive Neural Network Learning Algorithms for Multi-Category Real-Valued Pattern Classification.* Tech. rept. ISU-CS-TR97-06. Department of Computer Science, Iowa State University.

Parekh, R., Yang, J., & Honavar, V. (1997b). **MUpstart** - A Constructive Neural Network Learning Algorithm for Multi-Category Pattern Classification. Pages 1924–1929 of: *Proceedings of the IEEE/INNS International Conference on Neural Networks (ICNN-97).*

Parekh, R., Yang, J., & Honavar, V. (1997c). Pruning Strategies for Constructive Neural Network Learning Algorithms. Pages 1960–1965 of: *Proceedings of the IEEE/INNS International Conference on Neural Networks (ICNN-97).*

Parker, D. (1985). *Learning Logic.* Tech. rept. Technical Report TR-47. Center for Computational Research in Economics and Management Science, MIT. Cambridge. MA.

Pawlak, Z. (1991). *Rough Sets, Theoretical Aspects of Reasoning about Data.* NY: Kluwer Academic.

Porter, M. (1980). An Algorithm for Suffix Stripping. *Program*, 14(3). 130–137.

Post. E. (1943). Formal Reductions of the General Combinatorial Decision Problem. *Amer. J. Math.*, **65**, 197–215.

Poulard. H. (1995). Barycentric Correction Procedure: A Fast Method of Learning Threshold Units. Pages 710–713 of: *Proceedings of WCNN-95, July 17-21. Washington D.C.*, vol. 1.

Powell, M. (1987). Radial Basis Functions for Multivariable Interpolation: A Review. Pages 143–167 of: Mason, J., & Cox, M. (eds), *Algorithms for Approximation.* Oxford: Clarendon Press.

Punch, W., Goodman, E., Pei, M., Chia-Shun, L., Hovland, P., & Enbody, R. (1993). Further Research on Feature Selection and Classification Using Genetic Algorithms. Pages 557–564 of: *Proceedings of the International Conference on Genetic Algorithms.* NY: Springer.

Quinlan, R. (1986). Induction of Decision Trees. *Machine Learning*, **1**, 81–106.

Quinlan, R. (1993). *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann.

Raffin, B., & Gordon, M. (1995). Learning and Generalization with Minimerror. A Temperature-Dependent Learning Algorithm. *Neural Computation*. **7**, 1206–1224.

Reed, R. (1993). Pruning Algorithms — A Survey. *IEEE Transactions on Neural Networks*. 4(5), 740–747.

Richards, B., & Mooney, R. (1995). Automated Refinement of First-Order Horn-Clause Domain Theories. *Machine Learning*, **19**, 95–131.

Richeldi, M., & Lanzi, P. (1996). Performing Effective Feature Selection by Investigating the Deep Structure of the Data. Pages 379–383 of: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. Boston, MA: AAAI Press.

Ripley, B. (1996). *Pattern Recognition and Neural Networks*. New York: Cambridge University Press.

Rissanen, J. (1978). Modelling by Shortest Data Description. *Automatica*. **14**, 465–471.

Rosenblatt, F. (1958). The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*. **65**, 386–408.

Rosenschein, J., & Zlotkin, G. (1994). *Rules of Encounter: Designing Conventions for Automated Negotiation Among Computers*. Cambridge, MA: MIT Press.

Roy, A., Govil, S., & Miranda, R. (1995). An Algorithm to Generate Radial Basis Function (RBF)-Like Nets for Classification Problems. *Neural Networks*. 8(2). 179–201.

Rumelhart, D., Hinton, G., & Williams, R. (1986). Learning Internal Representations by Error Propagation. In: *Parallel Distributed Processing: Explorations into the Microstructure of Cognition*, vol. 1 (Foundations). Cambridge, Massachusetts: MIT Press.

Russell, S., & Norvig, P. (1995). *Artificial Intelligence: A Modern Approach.* Englewood Cliffs. NJ: Prentice Hall.

Saffery, J., & Thornton, C. (1991). Using Stereographic Projection as a Preprocessing Technique for Upstart. Pages 441–446 of: *Proceedings of the International Joint Conference on Neural Networks.* Piscataway, NJ: IEEE Press.

Salton, G. (1982). *Introduction to Modern Information Retrieval.* New York: McGraw-Hill.

Salton, G. (1989). *Automatic Text Processing: the Transformation, Analysis, and Retrieval of Information by Computer.* Reading, Massachusetts: Addison-Wesley.

Salton, G., & McGill, M. (1983). *Introduction to Modern Information Retrieval.* New York: McGraw Hill.

Sandholm, T. (1993). An Implementation of the Contract Net Protocol Based on Marginal Cost Calculations. Pages 295–308 of: *12th International Workshop on Distributed Artificial Intelligence.*

Sandholm, T. (1997). Necessary and Sufficient Contract Types for Optimal Task Allocation. In: *Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-97).*

Sandholm, T. (1998). Contract Types for Satisficing Task Allocation: I Theoretical Results. In: *AAAI 1998 Spring Symposium on Satisficing Models.*

Schlimmer, J. (1993). Efficiently Inducing Determinations: A Complete and Systematic Search Algorithm that Uses Optmal Pruning. Pages 284–290 of: *Proceedings of the Tenth International Conference on Machine Learning.* San Mateo, CA: Morgan Kaufmann.

Shavlik, J. W. (1994). A Framework for Combining Symbolic and Neural Learning. Pages 561–580 of: Honavar, V., & Uhr, L. (eds), *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration.* New York: Academic Press.

Sheinvald, J., Dom, B., & Niblack, W. (1990). A Modelling Approach to Feature Selection. Pages 535–539 of: *Proceedings of the Tenth International Conference on Pattern Recognition.*

Sheth, A., & Larson, J. (1990). Federated Databases: Architectures and Issues. *ACM Computing Surveys*, **22**(3), 183–236.

Shynk, J. J. (1990). Performance Surfaces of a Single-layer Pereptron. *IEEE Transactions on Neural Networks*, **1**, 268–274.

Siedlecki, W., & Sklansky, J. (1988). On Automatic Feature Selection. *International Journal of Pattern Recognition*, **2**, 197–220.

Siedlecki, W., & Sklansky, J. (1989). A Note on Genetic Algorithms for Large-scale Feature Selection. *IEEE Transactions on Computers*, **10**, 335–347.

Siu, K-Y., Roychowdhury, V., & Kailath, T. (1995). *Discrete Neural Computation - A Theoretical Foundation.* Englewood Cliffs, NJ: Prentice-Hall.

Skalak, D. (1994). Prototype and Feature Selection by Sampling and Random Mutation Hill-climbing Algorithms. Pages 293–301 of: *Proceedings of the Eleventh International Conference on Machine Learning.* San Mateo, CA: Morgan Kaufmann.

Smith, R. (1980). The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers*, **C-29**(12), 1104–1113.

Stanfill, C., & Waltz, D. (1986). Toward Memory-Based Reasoning. *Communications of the ACM*, **29**(12), 1213–1228.

Thrun, S. (1995). *Lifelong Learning: A Case Study.* Tech. rept. CMU-CS-95-208. Carnegie Mellon University, Pittsburgh, PA.

Towell, G., & Shavlik, J. (1993). Extracting Rules from Knowledge-Based Neural Networks. *Machine Learning*, **13**, 71–101.

Towell, G., & Shavlik, J. (1994). Knowledge-based Artificial Neural Networks. *Artificial Intelligence*, **70**(1–2), 119–165.

Towell, G., Shavlik, J., & Noordwier, M. (1990). Refinement of Approximate Domain Theories by Knowledge-Based Neural Networks. Pages 861–866 of: *Proceedings of the Eighth National Conference on Artificial Intelligence.*

Turing, A. (1936). On Computable Numbers with an Application to the Entscheidungsproblem. *Proc. London Math. Soc.*, **2**(42), 230–265.

Turney, P. (1994). Theoretical Analyses of Cross-Validation Error and Voting in Instance-Based Learning. *Journal of Experimental and Theoretical Artificial Intelligence*, **6**. 331–360.

Tversky, A. (1977). Features of Similiarity. *Psychological Review*, **84**. 327–352.

Uhr, L. (1984). *Algorithm-Structured Computer Arrays and Networks: Architectures and Processes for Images, Percepts, Models, Information.* New York: Academic Press.

Uhr, L., & Honavar, V. (1994). Introduction. In: Honavar, V., & Uhr, L. (eds), *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration.* Academic Press: New York.

Vafaie, H., & De Jong, K. (1993). Robust Feature Selection Algorithms. Pages 356–363 of: *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence.*

Valiant, L. (1984). A Theory of the Learnable. *Communications of the ACM*, **27**, 1134–1142.

Weigend, A., Rumelhart, D., & Huberman, B. (1991). Generalization by Weight-Elimination with Application to Forecasting. Pages 875–882 of: Lippmann, R., Moody, J., & Touretzky, D. (eds), *Advances in Neural Information Processing Systems.* San Mateo, CA: Morgan Kaufmann.

Weiss, S., & Kapouleas, I. (1989). An Empirical Comparison of Pattern Recognition, Neural Nets, and Machine Learning Classification Methods. Pages 781–787 of: *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence.*

Werbos, P. (1974). *Beyond Regression: New Tools for Prediction and Analysis in Behavioral Sciences.* Ph.D. thesis, Harvard University, Cambridge, MA.

Wettschereck, D., Aha, D., & Mohri, T. (1995). *A Review and Empirical Evaluation of Feature Weighting Methods for a Class of Lazy Learning Algorithms.* Tech. rept. AIC95-012. Naval Research Laboratory, Navy Center for Applied Research in Artificial Intelligence. Washington, D.C.

White, J. (1997). Mobile Agents. In: Bradshaw, J. (ed), *Software Agents.* Cambridge, MA: MIT Press.

Widrow, B., & Hoff, M. (1960). Adaptive Switching Circuits. *Institute of Radio Engineers. Western Electronic Show and Convention. Convention Record, part 4,* 96–104.

Wiederhold, G. (1997). The Conceptual Basis for Mediation Services. *IEEE Expert.* **12**(5). 38–46.

Wilson, D., & Martinez, T. (1997). Improved Heterogeneous Distance Functions. *Journal of Artificial Intelligence Research.* **6**. 1–34.

Wong, J., Honavar, V., Miller, L., & Naganathan, V. (1999). *Design and Implementation of Mobile Agent Infrastructure Based on Mobile Agent Interoperability Facilities (MAF).* Submitted for publication.

Yang, J., & Honavar, V. (1991). Experiments with the Cascade-Correlation Algorithm. Pages 369–380 of: *Proceedings of the 4th UNB AI Symposium.*

Yang, J., & Honavar, V. (1996). A Simple Randomized Quantization Algorithm for Neural Network Pattern Classifiers. Pages 223–228 of: *Proceedings of the World Congress on Neural Networks.*

Yang, J., & Honavar, V. (1997). Feature Subset Selection Using a Genetic Algorithm. Pages 380–385 of: *Proceedings of the Genetic Programming Conference (GP-97).*

Yang, J., & Honavar, V. (1998a). Feature Subset Selection Using a Genetic Algorithm. *IEEE Intelligent Systems (Sepcial Issue on Feature Transformation and Subset Selection)*, **13**, 44–49.

Yang, J., & Honavar, V. (1998b). Feature Subset Selection Using a Genetic Algorithm. In: *Feature Extraction, Construction and Selection - A Data Mining Perspective*. NY: Kluwer Academic.

Yang, J., Parekh, R., & Honavar, V. (1996). MTiling - A Constructive Neural Network Learning Algorithm for Multi-Category Pattern Classification. Pages 182–187 of: *Proceedings of the World Congress on Neural Networks.*

Yang, J., Havaldar, R., Honavar, V., Miller, L., & Wong, J. (1998a). Coordination of Distributed Knowledge Networks Using Contract Net Protocol. Pages 71–74 of: *Proceedings of the IEEE Information Technology Conference.*

Yang, J., Parekh, R., & Honavar, V. (1998b). DistAl: An Inter-pattern Distance-based Constructive Learning Algorithm. Pages 2208–2213 of: *Proceedings of the International Joint Conference on Neural Networks.*

Yang, J., Honavar, V., Miller, L., & Wong, J. (1998c). Intelligent Mobile Agents for Information Retrieval and Knowledge Discovery from Distributed Data and Knowledge Sources. Pages 99–102 of: *Proceedings of the IEEE Information Technology Conference.*

Yang, J., Pai, P., Honavar, V., & Miller, L. (1998d). Mobile Intelligent Agents for Document Classification and Retrieval: A Machine Learning Approach. Pages 707–712 of: *Proceedings of the 14th European Meeting on Cybernetics and Systems Research: Symposium on Agent Theory to Agent Implementation.*

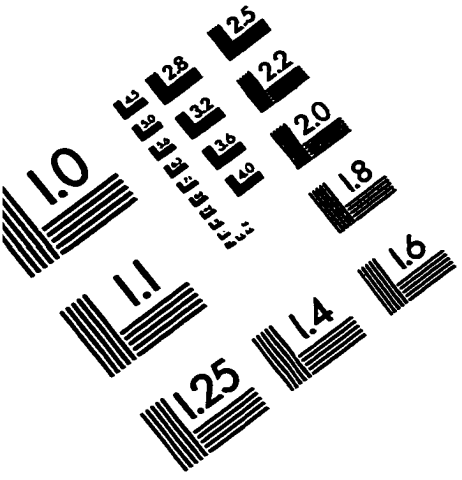
Yang, J., Parekh, R., & Honavar, V. (1999a). DistAl: An Inter-pattern Distance-based Constructive Learning Algorithm. *Intelligent Data Analysis.* To appear.

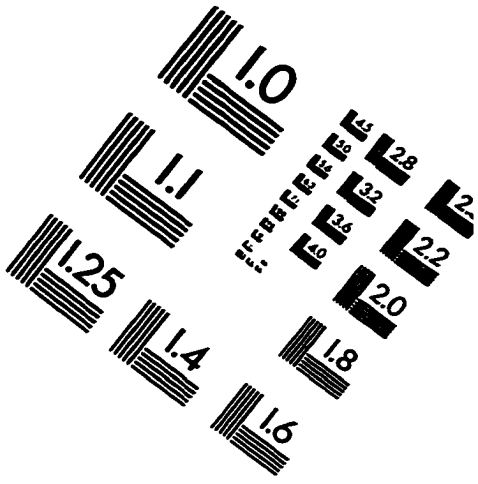
Yang, J., Parekh, R., & Honavar, V. (1999b). Theory Refinement Using a New Constructive Neural Network Learning Algorithm. In: *Proceedings of the International Joint Conference on Neural Networks.* To appear.

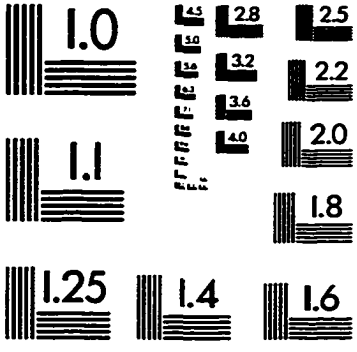
Zalzala, A., & Morris, A. (1996). *Neural Networks for Robotic Control: Theory and Applications.* New York: Ellis Horwood.

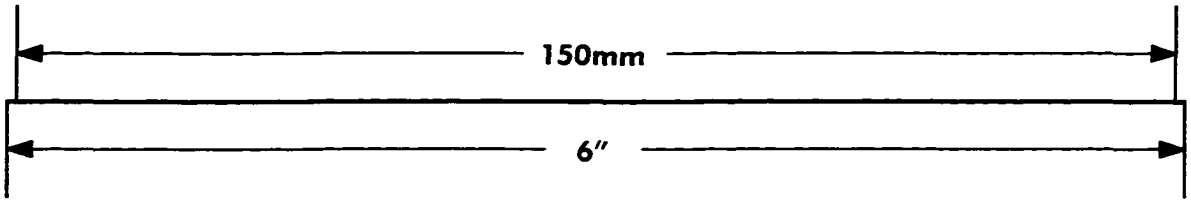
Zhou, G., McCalley, J., & Honavar, V. (1997). Power System Security Margin Prediction Using Radial Basis Function Networks. In: *Proceedings of the 29th Annual North American Power Symposium.*

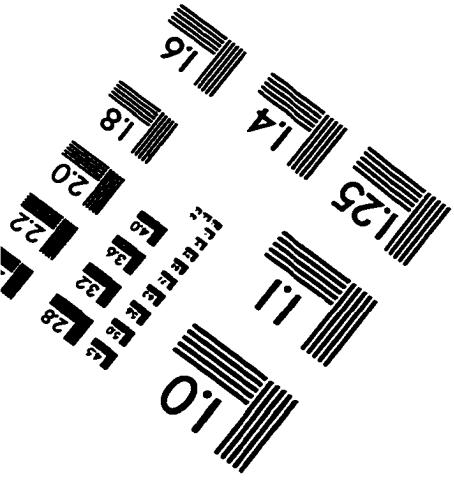
IMAGE EVALUATION
TEST TARGET (QA-3)

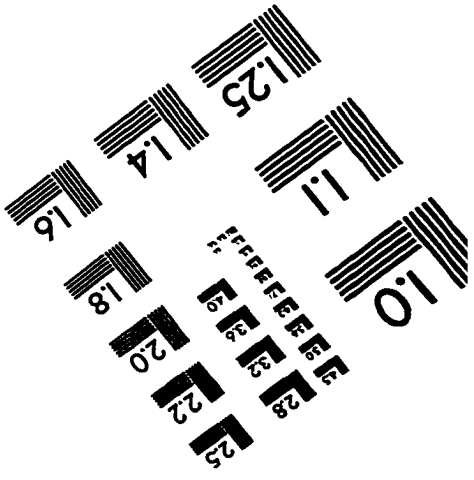












APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989